LUPA SEAL

Testing js Frameworks

Front-end JS Frameworks

Back-end JS Frameworks

JavaScripts Frameworks

Realvasi

php

HTML CSS JS

Jinja + Flask

AJAX

jQuery

{JSON}

```
h{
scroll-behavior: smooth;

background: greenyellow;
background: linear-gradient(to right, skyblue, lightgreen);
background-color:grey;
/*adding background image*/
background-image:url(/images/background.png);
/*will not repeat when reached last element*/
background-repeat: no-repeat;
/*will not move with scroll*/
background-attachment:fixed;
background-position: center;
/*image will resize dynamically*/
background-size: cover;

/*fonts.google.com*/
font-family: "consolas", sans-serif;
font-style: italic;
font-weight: bold;
font-size: 18px;
font-size: large;
color: white;

/*static,relative,absolute,sticky*/
position:relative;
top: 70px;
left: 30px;
/*will remain same position when scrolling*/
position:fixed;
/*will stick to bottom or top whn scrolling */
position:sticky;
/*becomes like a ghost, document will ignore this*/
position: absolute;
/*use clear:right; to clear it*/
float:right;

/*margin is the space around an element*/ write like this instead of top,bottom,right,left*/
margin: 0px;
/* use % to resize dynamically*/
margin-top: 50%;
margin-left: 15px;
margin-bottom: 20px;
margin-right: 50px;
/*this will remain left when resizing window*/
margin-left=auto;

/*padding is the space between the content and the border*/
padding: 50px;

border-top-style:solid;
border-bottom-style: double;
border-left-width: 10px;
```

```
  border-left-color: silver;
  border-width: 5px;
  border-color: gold;
  border-radius: 10px;
  border: 5px solid;
  border-color:blue;
  width: 250;
  height:250;

  text-decoration: cyan dotted underline;
  text-decoration: none;
  text-align:center;
  }

 a: hover{
    text-decoration: underline;
  }
```

```
//navbar
//make the navbar be fixed when scrolling down: fixed-top
//make distance from top: py-3
<nav class="navbar navbar-expand-lg navbar-light bg-light py-3 fixed-top">
```

```
// Printing
document.write("<h1>Hello world<h1></h1>");
document.write("<hr>");
document.write("<p>This is a javascript tutorial</p>");
alert("This is an alert");
console.log("logging to the console;")

//Variables
name="john";
document.write("your name is " + name + "<br>");
var age = 20;

//Casting & Converting
document.write( 100 + Number("25") + "<br>");
document.write( 100 + parseInt("50") + "<br>");
document.write( 100 +parseFloat("50.00") + "<br>");

//Strings
var greeting = "Hello";
```

```
    indexes:    01234
document.write( greeting.length + "<br>");
document.write( greeting.charAt(0) + "<br>" );
document.write( greeting.indexOf("llo") + "<br>");
//substring from position 2
document.write( greeting.substring(2) + "<br>");
//substring between 1 and 3
document.write( greeting.substring(1,3) + "<br>");

//Numbers
// Basic Arithmetic
document.write( 2 * 3 + "<br>");
// Exponents
document.write( 2 ** 3 + "<br>");
// Modulus
document.write( 10 % 3 + "<br>");
//Order of operations
document.write( (1 + 2) * 3 + "<br>");
//int's and doubles
document.write( 10 / 3.0 + "<br>");
var num = 10;
num+=100; // +=, -=, /=, *=
document.write(num + "<br>");
num++;
document.write(num + "<br>");
num++;
document.write(num + "<br><br>");

//Math class has useful math methods
document.write( Math.pow(2, 3) + "<br>");
document.write( Math.sqrt(144) + "<br>");
document.write( Math.round(2.7) + "<br>");

//User Input
var name = window.prompt("Enter your name: ");
alert("Your name " + name);

//Getting HTML
//get element by id
var header = document.getElementById("myHeader");
// add css to the element
header.style="color:blue; background-color:red;"
//print attribute
document.write( header.getAttribute("giraffe"));
//replace element with something else
header.innerHTML = "Elephant Academy";

//Arrays
var luckyNumbers = []
var luckyNumbers= [4, 8, 15, 16, "twenty", " false"];
    //indexes:        0 1   2   3       4            5
luckyNumbers[0] = 90;
document.write(luckyNumbers[0] + "<br>");
document.write(luckyNumbers[1] + "<br>");
document.write(luckyNumbers.length);

//N Dimensional Arrays
var numberGrid = [ [1, 2], [3, 4] ];
numberGrid[0][1] = 99;
document.write(numberGrid[0][0] + "<br>");
document.write(numberGrid[0][1] + "<br>");
```

```
//Array functions
var friends = new Array();
friends.push("Oscar");
friends.push("Angela");
friends.push("Kevin");
    friends.pop()
    document.write( friends + "<br>");
    document.write( friends.indexOf("Angela") + "<br>");
    document.write( friends.indexOf("Z") + "<br>");
    document.write( friends.reverse("Angela") + "<br>");
    document.write( friends.sort() + "<br>");

//Objects
var student= {
  name: "Jim",
  major: "Business",
  age: 19,
  gpa: 2.5
};
student.name = "Andy"
document.write( student.name + "<br>" );
document.write( student.major + "<br>" );
document.write( student.gpa + "<br>");

//Functions
var sum = addNumbers(4, 60);
document.write(sum);
function addNumbers(num1, num2){
  return num1 + num2
}
// Inline event listener with id & event handler #1st method
function handleClick(element){
  alert("Clicked " + element.id);
}
// Inline event listener only with id from HTML stored in a var and you drop the on from onclick #2nd method
var header = document.getElementById("myHeader");
header.addEventListener("click", function(){
  alert("You clicked " + header.id);
});

//If statements
var isStudent = false;
var isSmart = false;
// Logical (boolean)operators: &&, ||, !
if (isStudent && isSmart){
  document.write("You are a student");
} else if (isStudent && !isSmart){
  document.write("You are not a smart student");
} else {
  document.write("You are not a student and not smart");
}
document.write("<br>");
// Comparison Operators: >, <, >=, !=, ==,===, String.equals()
if(1 < 3){
  document.write("number comparison was true");
  document.write("<br>");
}
if("dog" == "dog"){
  document.write("string comparison was true");
}

//Switch statements
```

```
var myGrade = "A";
switch(myGrade){
  case "A":
    document.write("You Pass");
    break;
  case "F":
    document.write("You Fail");
    break;
  default:
    document.write("Invalid grade");
}

//While loops
//check condition first then execute block of code
var index=1;
while(index <=5){
  document.write(index);
  index++;
}
//execute one time then execute block of code
do{
  document.write(index);
  index++;
}while(index <=5);

//For loops
for(var i=0; i < 5; i++){
  document.write(i);
}
//For loop over structures like an array(luckyNum will be the parameter)
var luckyNums = [4, 8, 15, 16, 23, 42];
luckyNums.forEach(function(luckyNum){
  document.write(luckyNum + "<br>");
});

//Exception catching
try{
  throw "something went wrong <br>"
  var x = y + 9;
}catch(err){
  document.write(err)
}finally{
  //this code always gets executed
  document.write('try with a different input')
}

//Classes & Objects
class Book{
    constructor(title, author){
        this.title = title;
        this.author = author;
  }
  //you do not need to put the word function
  readBook(){
        document.write("Reading " + this.title + " by " + this.author);
  }
}
var book1 = new Book("Harry Potter", "JK Rowling");
document.write(book1.title + "<br>");
book1.readBook();

//Getters & setters
```

```
class Book{
   constructor(title, author){
      this.title = title;
      this.author = author;
  }
  get title(){
     document.write("<p>getting</p>");
     return this._title;
  }
  set title(title){
     document.write("<p>setting</p>");
     this._title = title;
  }
  readBook(){
     document.write("Reading " + this.title + " by " + this.author);
  }
}
var book1 = new Book("Harry Potter", "JK Rowling");
document.write(book1.title + "<br>");
book1.readBook();

//Inheritance
class Chef{
  constructor(name, age){
      this.name = name;
      this.age = age;
  }
  makeChicken(){
      document.write("The chef makes chicken <br>");
  }
  makeSalad(){
      document.write("The chef makes salad <br>");
  }
  makeSpecialDish(){
      document.write("The chef makes a special dish <br>");
  }
}
class ItalianChef extends Chef{
  constructor(name, age, countryOfOrigin){
     super(name, age);
     this.countryOfOrigin = countryOfOrigin;
  }
  makePasta(){
     document.write("The chef makes pasta <br>");
  }
  //overridden
  makeSpecialDish(){
     document.write("The chef makes chicken parm <br>");
  }
}
var myChef = new Chef("Gordon Ramsay", 50);
myChef.makeChicken();
myChef.makeSpecialDish();
var myItalianChef = new ItalianChef("Massimo Bottura", 55, "Italy");
myItalianChef.makeChicken();
document.write(myItalianChef.age);
myItalianChef.makeSpecialDish();
```

```php
<?php

//Printing
echo "<h1>Hello World</h1>";
print "<hr>";
echo "<p>This is php tutorial</p>";

//Variables
$name = "Mike"; // Strings
$age = 30; // Integer
$gpa = 3.5; //Decimal
$is_tall = true;//Boolean->true/false
$name = "John";
echo "Your name is $name <br>";

//Casting & converting
echo ( (int)3.14 )."<br>";
echo ( (int)"80" + (float)"60.5" )."<br>";
echo intval("80") +floatval("60.5");

//Strings
$greeting = "Hello";
 //indexes:   01234
echo strlen($greeting)."<br>";
echo $greeting[0]."<br>";
echo $greeting[-1]."<br>";
echo str_replace("l","Z", $greeting."<br>");
echo strchr($greeting, "ll")."<br>";

//Numbers
//Basic arithmetic: +, -, /, *
echo (2 * 3)."<br>";
//Basic arithmetic: +, -, /, *
echo (2 **3)."<br>";
//Modulus Op.: return remainder of 10/
echo (10 % 3)."<br>";
//Order of operations
echo ((1 + 2) * 3)."<br>";
//int's and doubles
echo (10 / 3.0)."<br><br>";
$num = 10;
// +=, -=, /=, *=
$num += 100;
$num++;
echo $num."<br><br>";
//Useful math methods
echo max(2, 3)."<br>";
echo sqrt(144)."<br>";
```

```
echo round(2.7)."<br>";

//User input – GET
//stores information in the URL
echo $_GET["username"];
echo "</br>";
$age = $_GET["age"];
echo $age;
// User input - POST - more secure way to store information
echo $_POST["username"];

//Lists
$lucky_numbers = [];
$lucky_numbers = array(4, 8, "fifteen", 16, 23, 42.0);
$lucky_numbers = [4, 8, "fifteen", 16, 23, 42.0];
  //indexes              0  1      2         3   4     5
$lucky_numbers[0] = 90;
echo $lucky_numbers[0]."<br>";
echo $lucky_numbers[1]."<br>";
echo count($lucky_numbers)."<br>";
//N dimnesional lists
$number_grid =[ [1,2], [3,4] ];
$number_grid[0][1] = 99;
echo $number_grid[0][0]."<br>";
echo $number_grid[0][1]."<br>";

//List functions
$friends =[];
array_push($friends, "Oscar", "Angela");
array_push($friends, "Kevin");
array_pop($friends);
echo "$friends[0], $friends[1], $friends[2] <br>";
sort($friends);
echo "$friends[0], $friends[1], $friends[2] <br>";
echo in_array("Oscar", $friends);

//Associative arrays
$testGrades = array(
   "Andy" => "B+",
   "Stanley" => "C",
   "Ryan" => "A",
   3 => 95.2
);
echo $testGrades["Andy"]."<br>";
echo $testGrades["Ryan"]."<br>";
echo $testGrades[3]."<br>";

//Functions
function addNumbers($num1, $num2=99){
   return $num1 + $num2;
}
$sum = addNumbers(4, 3);
echo $sum;

//If statements
$isStudent = false;
$isSmart = false;
if($isStudent && $isSmart){
   echo "You are a student";
}elseif($isStudent || !$isSmart){
   echo "You are not a smart student";
} else {
```

```
    echo "You are not a student and not smart";
}
echo "<br>";
//>, <, >=, !=, ==
if(1 < 3){
    echo "number comparison was true";
}
echo "<br>";
if("dog" == "dog"){
    echo "string comparison was true";
}

//Switch statements
$myGrade = "A";
switch($myGrade){
    case "A":
        echo "You Pass";
        break;
    case "F":
        echo "You  fail";
        break;
    default:
    echo "Invalid grade";
}

//While loop
$index = 1;
while ($index <=5){
    echo $index;
    $index += 1;
}
do{
    echo $index;
    $index += 1;
}while ($index <=5);

//For loops
for($i = 0; $i < 5; $i++){
    echo $i;
}
$luckyNums = [4, 8, 15, 23, 42];
foreach($luckyNums as $luckyNum){
    echo $luckyNum."<br>";
}

//Exception catching
try{
    throw new Exception('Something bad happened');
} catch(Exception $e){
    echo $e->getMessage();
} finally{
    echo "<br> This code gets executed no matter what";
}

//Classes & Objects
class Book{
    var $title;
    public $author;
    public static $staticAttribute = "My Static Attribute";
    function readBook(){
        echo "Raading $this->title by $this->author";
    }
```

```
};
$book1 = new Book;
$book1->title = "Harry Potter";
$book1->author = "JK Rowling";
echo $book1->title. "<br>";
echo Book::$staticAttribute. "<br>";
$book1->readBook();

//Constructors
class Book{
   var $title;
   public $author;
   function __construct($title, $author){
      $this->title = $title;
      $this->author = $author;
   }
   function readBook(){
      echo "Reading $this->title by $this->author";
   }
};
$book1 = new Book("Harry Potter", "JK Rowlling");
$book1->title ="Half-Blood Prince";
echo $book1->title."<br>";a

//Getters & Setters
class Book{
   private $title;
   public $author;
   function __construct($title, $author){
      $this->setTitle($title);
      $this->author = $author;
   }
   function getTitle(){
      return $this->title;
   }
   function setTitle($title){
      $this->title = $title;
   }
   function readBook(){
      echo "Reading $this->title by $this->author";
   }
};
$book1 = new Book("Harry Potter", "JK Rowlling");
$book1->setTitle("Half-Blood Prince");
echo $book1->getTitle();

//Inheritance
class Chef{
   function makeChicken(){
      echo "The chef makes chicken";
   }
   function makeSalad(){
      echo "The chef makes salad";
   }
   function makeSpecialDish(){
      echo "The chef makes bbq ribs";
   }
};
class ItalianChef extends Chef{
   function makePasta(){
      echo "The chef makes pasta";
   }
```

```
    function makeSpecialDish(){
        echo "The chef makes chicken parm";
    }
};
$chef = new Chef();
$chef->makeChicken();
echo "<br>";
$italianChef = new ItalianChef();
$italianChef->makeChicken();
echo "<br>";
$chef->makeSpecialDish();
echo "<br>";
$italianChef->makeSpecialDish();
```

```
//Inheritance extended
    class Chef{
        public $name;
        public $age;
        function __construct($name, $age){
            $this->name = $name;
            $this->age = $age;
        }
        function makeChicken(){
            echo "The chef makes chicken";
        }
        function makeSalad(){
            echo "The chef makes salad";
        }
        function makeSpecialDish(){
            echo "The chef makes bbq ribs";
        }
    };
    class ItalianChef extends Chef{
        public $countryOfOrigin;
        function __construct($name, $age, $countryOfOrigin){
            $this->countryOfOrigin = $countryOfOrigin;
            parent::__construct($name, $age);
        }
        function makePasta(){
            echo "The chef makes pasta";
        }
        function makeSpecialDish(){
            echo "The chef makes chicken parm";
        }
    };
    $chef = new Chef("Gordon Ramsay", 50);
    $chef->makeChicken();
    echo "<br>";
    $italianChef = new ItalianChef("Massimo Bottura", 55, "Italy");
    $italianChef->makeChicken();
    echo "<br> $italianChef->age";
```

```
//Abstract classes & attributes
abstract class Vehicle{
    public abstract function move();
    public function getDescription(){
        echo "Vehicles are used for transportation";
    }
}
class Bicycle extends Vehicle{
    public function move(){
        echo "The bicycle pedals forward";
```

```
    }
}
class Plane extends Vehicle{
    public function move(){
        echo "The plane flys through the sky";
    }
}
$plane = new Plane();
$plane->move();
echo "<br>";
$plane->getDescription();

//Interface inheritance (polymorphism)
interface Animal{
    public function speak();
}
class Cat implements Animal{
    public function speak(){
        echo "Mewo Mewo <br>";
    }
}
class Dog implements Animal{
    public function speak(){
        echo "Woof Woof <br>";
    }
}
$animals = [
    new Dog(),
    new Cat()
];
foreach($animals as $animal){
    $animal->speak();
}
```

ubuntu: cd /mnt/c

```
$ mkdir froshims
$ cd froshims/
froshims/ $ mkdir templates
froshims/ $ code app.py
```

```
pip3 install virtualenv
virtualenv env
source env/bin/activate
pip3 install flask
python3 app.py
deactivate
```

```
froshims/ $ code templates/index.html
```

```
pip install cs50
```

```
pip install Flask-Session
```

Main form

app.py
```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, test"

if __name__ == "__main__":
    app.run(debug=True)
```

Input from HTML -> navigate to different page -> use layout template

app.py
```
@app.route("/", methods=["GET","POST"])
def index():
    if request.method =="GET":
        return render_template("index.html")
    elif request.method == "POST":
        return render_template("greet.html", name=request.form.get("name", "world"))
```

layout.html
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    % block body %}{% endblock %}
</body>
</html>
```

index.html
```
{% extends "layout.html" %}
{% block body %}
    <form action="/" method="post">
        <input autocomplete ="off" autofocus placeholder="Name" name="name" type="text">
        <button type="submit">Greet</button>
    </form>
{% endblock %}
```

greet.html
```
{% extends "layout.html" %}
{% block body %}
    hello, {{ name }}
{% endblock %}
```

Input from HTML -> navigate to different page -> use layout template -> use global dictionary to store info and global list to use as drop down option

app.py
```
from flask import Flask, render_template, request

app = Flask(__name__)

REGISTRANTS ={}

SPORTS = [
    "Basketball",
    "Soccer",
    "Ultimate Fribee"
]
```

```python
@app.route("/")
def index():
    return render_template("index.html", sports=SPORTS)

@app.route("/register", methods=["POST"])
def register():
    name = request.form.get("name")
    if not name:
        return render_template("failure.html")
    sport = request.form.get("sport")
    if sport not in SPORTS:
        return render_template("failure.html")
    REGISTRANTS[name] = sport
    return render_template("success.html")

@app.route("/registrants")
def registrants():
    return render_template("registrants.html", registrants=REGISTRANTS)

if __name__ == "__main__":
    app.run(debug=True)
```

layout.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    % block body %}{% endblock %}
</body>
</html>
```

index.html

```
{% extends "layout.html" %}
{% block body %}
    <form action="/register" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" required type="text">
        <select name="sport">
            <option disabled selected>Sport</option>
        {% for sport in sports %}
            <option value="{{ sport }}">{{sport}}</option>
        {% endfor %}
        </select>
        <button type="submit">Register</button>
    </form>
{% endblock %}
```

success.html

```
{% extends "layout.html" %}
{% block body %}
    You are registered!
{% endblock %}
```

failure.html

```
{% extends "layout.html" %}
{% block body %}
    You are not registered!
{% endblock %}
```

registrants.html

```
{% extends "layout.html" %}
{% block body %}
    <ul>
        {% for name in registrants %}
        <li>{{ name }} is registered for {{ registrants[name]}} </li>
```

```
        {% endfor %}
    </ul>
{% endblock %}
```

Implements a registration form, storing registrants in a SQLite database, with support for deregistration:

app.py

```
from cs50 import SQL
from flask import Flask, redirect, render_template, request

app = Flask(__name__)

db = SQL("sqlite:///froshims.db")

SPORTS = [
    "Basketball",
    "Soccer",
    "Ultimate Frisbee"
]

@app.route("/")
def index():
    return render_template("index.html", sports=SPORTS)

@app.route("/deregister", methods=["POST"])
def deregister():

    # Forget registrant
    id = request.form.get("id")
    if id:
        db.execute("DELETE FROM registrants WHERE id = ?", id)
    return redirect("/registrants")

@app.route("/register", methods=["POST"])
def register():

    # Validate submission
    name = request.form.get("name")
    sport = request.form.get("sport")
    if not name or sport not in SPORTS:
        return render_template("failure.html")

    # Remember registrant
    db.execute("INSERT INTO registrants (name, sport) VALUES(?, ?)", name, sport)

    # Confirm registration
    return redirect("/registrants")

@app.route("/registrants")
def registrants():
    registrants = db.execute("SELECT * FROM registrants")
    return render_template("registrants.html", registrants=registrants)

if __name__ == "__main__":
    app.run(debug=True)
```

layout.html

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>froshims</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

index.html

```
{% extends "layout.html" %}

{% block body %}
    <h1>Register</h1>
    <form action="/register" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
        {% for sport in sports %}
            <input name="sport" type="radio" value="{{ sport }}"> {{ sport }}
        {% endfor %}
        <button type="submit">Register</button>
    </form>
{% endblock %}
```

registrants.html

```
{% extends "layout.html" %}

{% block body %}
    <h1>Registrants</h1>
    <table>
        <thead>
            <tr>
                <th>Name</th>
                <th>Sport</th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            {% for registrant in registrants %}
                <tr>
                    <td>{{ registrant.name }}</td>
                    <td>{{ registrant.sport }}</td>
                    <td>
                        <form action="/deregister" method="post">
                            <input name="id" type="hidden" value="{{ registrant.id }}">
                            <button type="submit">Deregister</button>
                        </form>
                    </td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

failure.html

```
{% extends "layout.html" %}

{% block body %}
    You are not registered!
{% endblock %}
```

requirements.txt

```
cs50
Flask
```

froshims.db

```
sqlite3
.schema
SELECT*FROM registrants
```

login page using flask:

app.py

```
from flask import Flask, redirect, render_template, request, session
from flask_session import Session

# Configure app
```

```python
app = Flask(__name__)

# Configure session
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

@app.route("/")
def index():
    if not session.get("name"):
        return redirect("/login")
    return render_template("index.html")

@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        session["name"] = request.form.get("name")
        return redirect("/")
    return render_template("login.html")

@app.route("/logout")
def logout():
    session["name"] = None
    return redirect("/")

if __name__ == "__main__":
    app.run(debug=True)
```

layout.html

```html
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>store</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

index.html

```
{% extends "layout.html" %}

{% block body %}

    {% if session["name"] %}
        You are logged in as {{ session["name"] }}. <a href="/logout">Log out</a>.
    {% else %}
        You are not logged in. <a href="/login">Log in</a>.
    {% endif %}

{% endblock %}
```

login.html

```
{% extends "layout.html" %}

{% block body %}

    <form action="/login" method="post">
        <input autocomplete="off" autofocus name="name" placeholder="Name" type="text">
        <button type="submit">Log In</button>
    </form>

{% endblock %}
```

requirements.txt

```
Flask
Flask-Session
```

Store

app.py

```
from cs50 import SQL
from flask import Flask, redirect, render_template, request, session
from flask_session import Session

# Configure app
app = Flask(__name__)

# Connect to database
db = SQL("sqlite:///store.db")

# Configure session
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
Session(app)

@app.route("/")
def index():
    books = db.execute("SELECT * FROM books")
    return render_template("books.html", books=books)

@app.route("/cart", methods=["GET", "POST"])
def cart():

    # Ensure cart exists
    if "cart" not in session:
        session["cart"] = []

    # POST
    if request.method == "POST":
        id = request.form.get("id")
        if id:
            session["cart"].append(id)
        return redirect("/cart")

    # GET
    books = db.execute("SELECT * FROM books WHERE id IN (?)", session["cart"])
    return render_template("cart.html", books=books)

if __name__ == "__main__":
    app.run(debug=True)
```

layout.html

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>store</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```

books.html

```
{% extends "layout.html" %}

{% block body %}

    <h1>Books</h1>
    {% for book in books %}
        <h2>{{ book["title"] }}</h2>
        <form action="/cart" method="post">
            <input name="id" type="hidden" value="{{ book['id'] }}">
            <button type="submit">Add to Cart</button>
        </form>
```

```
    {% endfor %}

{% endblock %}

cart.html
{% extends "layout.html" %}

{% block body %}

    <h1>Cart</h1>
    <ol>
        {% for book in books %}
            <li>{{ book["title"] }}</li>
        {% endfor %}
    </ol>

{% endblock %}

requirements.txt
cs50
Flask
Flask-Session

store.db
```

Searches for shows

```
app.py
from cs50 import SQL
from flask import Flask, render_template, request

app = Flask(__name__)

db = SQL("sqlite:///shows.db")

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/search")
def search():
    shows = db.execute("SELECT * FROM shows WHERE title LIKE ?", "%" + request.args.get("q") + "%")
    return render_template("search.html", shows=shows)

if __name__ == "__main__":
    app.run(debug=True)

layout.html
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>shows</title>
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>

index.html
{% extends "layout.html" %}

{% block body %}

    <form action="/search" method="get">
        <input autocomplete="off" autofocus name="q" placeholder="Query" type="search">
        <button type="submit">Search</button>
```

```
    </form>

{% endblock %}
```

search.html

```
{% extends "layout.html" %}

{% block body %}

    <ul>
        {% for show in shows %}
            <li>{{ show["title"] }}</li>
        {% endfor %}
    </ul>

{% endblock %}
```

requirements.txt

```
cs50
Flask
```

shows.db

Searches for shows using Ajax

app.py

```
from cs50 import SQL
from flask import Flask, render_template, request

app = Flask(__name__)

db = SQL("sqlite:///shows.db")

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/search")
def search():
    q = request.args.get("q")
    if q:
        shows = db.execute("SELECT * FROM shows WHERE title LIKE ? LIMIT 50", "%" + q + "%")
    else:
        shows = []
    return render_template("search.html", shows=shows)

if __name__ == "__main__":
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>shows</title>
    </head>
    <body>
```

```
        <input autocomplete="off" autofocus placeholder="Query" type="search">

        <ul></ul>

        <script>

            let input = document.querySelector('input');
            input.addEventListener('input', async function() {
                let response = await fetch('/search?q=' + input.value);
                let shows = await response.text();
                document.querySelector('ul').innerHTML = shows;
            });

        </script>

    </body>
</html>
```

search.html

```
{% for show in shows %}
    <li>{{ show["title"] }}</li>
{% endfor %}
```

requirements.txt

```
cs50
Flask
```

shows.db

ajax.js

```
function cs50Info(name)
{
    //deal with the situation nothing is chosen
    if(name == "")
        return;

    //create a new AJAX object
    var ajax = new XMLHttpRequest();

    //When the page is loaded, have a callback function pre-fill our div
    ajax.onreadystatechange = function(){
        if (ajax.readyState == 4 && ajax.status == 200) {
            $('#infodiv').html(ajax.resposeText);
        }
    };

    //open the requested file and transmit the data
    ajax.open('GET', name + '.html', true);
    ajax.send();
}
```

index.html

```
    <head>
    <title>Ajax</title>
    <script src="https://code.jquery.com/jquery-1.11.0.min.js" type="text/javascript"></script>
    <script src="js/ajax.js" type="text/javascript"></script>
    <link href="css/div.css" rel="stylesheet"/>
    </head>
    <body>
        <h2>Ajax Test</h2>
        <div id="infodiv">
            Information goes here
        </div>
        <div>
        <form>
            <select name="staff" onchange="cs50Info(this.value);">
                <option value="">Select someone:</option>
```

```
                <option value="bowden">Rob Bowden</option>
                <option value="chan">Zamyla Chan</option>
                <option value="malan">David J.</option>
                <option value="zlatkova">Maria Zlatkova</option>
            </select>
        </form>
```

chan.html

```
    <p>Zamyl Chan</p>
    <img src ="img/chan.jpg" />
    <p>Class of 2014</p>
    <p>Winthrop House</p>
```

//JSON = JavaScript Object Notation = language used to store information about various things
//We can store the information in key-value pairs, so these are things like variables, lists and objects
//We can use those different structures in order to structure our data and better define information about different things
//We write json by creating a json file.json
//access with person.name // person.age
//use loop through the list the hobbies to access the pieces of information

```
{
   "person": {
   "name":"Mike",
   "age" : 23,
   "gpa" :3.5,
   "fav_num": 1e+10,
   "male": true,
   "flaws": null,
   "hobbies": ["hiking","movies","riding bike"],
   "friends": [
     {
        "name" : "Steph",
        "age"  : 23
     },

     {
        "name" : "Adam",
        "age"  : 22
     }
   ]
 }
}
```

```
Searches for shows using Ajax with JSON

App.py
from cs50 import SQL
from flask import Flask, jsonify, render_template, request

app = Flask(__name__)

db = SQL("sqlite:///shows.db")

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/search")
def search():
    q = request.args.get("q")
    if q:
        shows = db.execute("SELECT * FROM shows WHERE title LIKE ? LIMIT 50", "%" + q + "%")
```

```
    else:
        shows = []
    return jsonify(shows)

if __name__ == "__main__":
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta name="viewport" content="initial-scale=1, width=device-width">
        <title>shows</title>
    </head>
    <body>

        <input autocomplete="off" autofocus placeholder="Query" type="text">

        <ul></ul>

        <script>

            let input = document.querySelector('input');
            input.addEventListener('input', async function() {
                let response = await fetch('/search?q=' + input.value);
                let shows = await response.json();
                let html = '';
                for (let id in shows) {
                    let title = shows[id].title.replace('<', '&lt;').replace('&', '&amp;');
                    html += '<li>' + title + '</li>';
                }
                document.querySelector('ul').innerHTML = html;
            });

        </script>

    </body>
</html>
```

requirements.txt

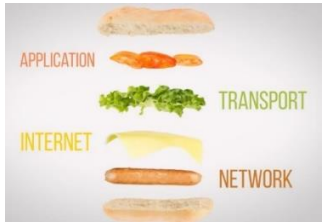
```
cs50
Flask
shows.db
```

//automatic use ! + tab

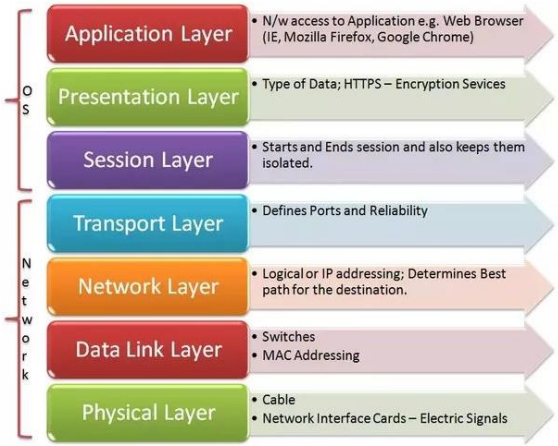
```
<!DOCTYPE html>
<html>
<head>
        //web bar title
        <title>Terminator</title>

        //Connect to css file
```

```
        <link rel="stylesheet" href="style.css">//add css external
        //Connect to js file
        <script src="js.js"></script>
        //Connect to Google Fonts
        <link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap" rel="stylesheet">//link fonts from google
        //Connect css to bootstrap
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
        //Connect to fontawesome
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.4/css/all.css" integrity="sha384-DyZ88mC6Up2uqS4h/KRgHuoeGwBcD4Ng9SiP4dIRy0EXTlnuz47vAwmeGwVChigm" crossorigin="anonymous"/>
    <meta charset="UTF-8">
    <meta name="description" content="T1000 against Skynet">
    <meta name="keywords" content="T3">
    <meta name="author" content="lupaseal">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    //Refresh page after 30'
    <meta http-equiv="refresh" content="30">

</head>
<body style="background-color: #07100e">
                                    <!-- -->//comments
                                    <!--span = adds markup to text or portion of a document-->
                                    <!--div = defines a division of a document-->
                                    <!--iframe = embed content from another source into HTML document, ex.used for ads, used in many hacking techniques-->
                                    <!--margin = space AROUND an element-->
                                    <!--padding = is the space between the content and the border-->
                                    <!--float = positions an element to the left or right side of a container, popular for wrapping elements around images-->


     //Headers
       <h1 style="color: #eeeee4">Terminator 3</h1> //word size h1->h6
       <header></header>
       <main></main>
       <footer> Copyright &#169 John Harvard </footer> // &#169 will print ©

     //add paragraph
        <p style="background-color:#ecdede;color:#edb879"><b><i><big><small><sub><sup><ins><del><mark>Lorem + tab, <span style="color:#b97455">testing</span></p></div>

    //horizontal line
       <hr>
    //line break//enter
       <br>

    //Add CSS inside HTML
       <div style=background-color:#ecdede">

    //add link
       <a href="https://www.econbridge.com" target=_self title="EconBridge is the way"> EconBridge </a>//href="pag2.html"//href=mailto:lupaseal@gmail.com
    //add image with link
       <a href="https://www.warnerbros.com/movies/terminator-3-rise-machines"><img src="\images\terminator3.png" height="250" width="400" alt="alternative text" title="terminator3"></a>

    //add image
       <img src="assets/imgs/logo.png">

    //add icon
       <i class="fa-solid fa-shopping-bag"></i>

    //add audio
      <audio controls src="\music\enjoy the silence.mp3"></audio>

    //add video
      <video controls src="terminatorvsrobocop.mp4" width="400"></video>
```

```
//add list
  <h4>Unordered List</h4><ul><li>Pizza</li><ul><li>Burgir</li></ul></ul>
  <h4>Ordered</h4><ol type="1"><li>eat</li></ol>
  <h4>Description List</h4><dl><dt>Terminator</dt><dd>Full movie</dd></dl>

//add table
  <table bgcolor="black" width="100">
    <tr bgcolor="grey"><th width="100">Terminators</th></tr>
    <tr bgcolor="lightgrey" align="center"><td>T1000</td></tr></table>

//web in a web
  <iframe style="border:0" src="https://www.econbridge.com" width="750" height="250">//src="add.html

//add button with link
  <a href="https://www.econbridge.com"><button style="background-color: #333333;color:#00FF00;border-radius:5px">click to go to econbrige</button></a>
//button with js functionality
  <button onclick="doSomething()" style="background-color: #333333;color:#00FF00;border-radius:5px">click to go to econbrige</button>
  <p id="test">Hello!</p>
  <script>function doSomething(){document.getElementById("test").innerHTML = "Goodbye";}</script>

//Forms
  <form action="action_page.php" method="POST">//action if you submit something
            <div><label for="fname">first name</label><input type="text" id="fname" name="fname" placeholder="terminator" required></div>
            <div><label for="pass">password</label><input type="password" id="pass" name="pass" placeholder="1234" maxlength="12" required></div>
            <div><label for="email">email</label><input type="email" id="email" name="email" placeholder="lupaseal@lupaseal.com"required></div>
            <div><label for="phone">phone</label><input type="tel" id="phone" name="phone" placeholder="(123)-450-7800"required></div>
            <div><label for="bdate">birth date</label><input type="date" id="bdate" name="bdate" required></div>
            <div><label for="quantity">quantity</label><input type="number" id="quantity" name="quantity" min="0" max="99" value="1" required></div>
            <div><label for="Mr.">Mr.</label>//radio buttons
                    <input type="radio" id="Mr." name="title" value="Mr.">
                <label for="Mrs.">Mrs.</label>
                    <input type="radio" id="Mrs."name="title" value="Mrs.">
                <label for="PHD.">PHD.</label>
                    <input type="radio" id="PHD."name="title" value="PHD."></div>
            <div><label for="payment">payment</label>//drop down
                    <select id="payment" name="payment"><option value="visa">Visa</option></select> </div>
            <div><label for="subscribe">subscribe</label>//checkbox
                    <input type="checkbox" id="subscribe" name="subscribe"></input></div>
            <div><input type="reset"></div>
            <div><input type="submit"></div>
  </form>
  //Add a box that will send the person to search over google
    <form action="https://www.google.com/search" method="get">
        <input autocomplete="off" autofocus name="q" placeholder="Query" type="text">
        <input type="submit">
  </form>

  //Position box inside a box so you will have 1st box relative and then 2nd one absolute
    <div id="box1"><div id="box2"></div></div>

 //Event handler with event #1st method
    <h1 id="myHeader" onclick="handleClick(this)">Giraffe Academy</h1>
//Event handler with id #2nd method
  <h1 id="myHeader">Giraffe Academy</h1>

 //Connect js to bootstrap
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>
</body>
</html>
```



A website is nothing but an application running on a remote server. It is the top layer in OSI model.



Now, when you say its IP address, its the IP of the server as websites don't have IP address. When you type the known IP address in your web browser, the browser will try to connect to port 80 of the remote web server. For the page to load anything, the remote server must run an web server on port 80 with a default page/index page stored in its web-apps directory.

- **IP Addresses are Logical Address:**
  An IP address is not a physical address that means *IP Address* of a device may change. The current *IP address* of your device on which you are reading this answer may change. once you go to Starbucks you connect your device to their *wi-fi router* more likely you will get a completely different *IP Address*. This happens every time when you connect your laptop/i pad with different network. You get different *IP Address*.
  So *IP Address* is a *Logical Address.*
- **Two versions of IP:**
  IPv4 & IPv6 :
  - **IPv4** (*Internet Protocol version 4) :*
    - We have been using it for long time.
    - *32 bit Binary Number* divided in *4 Octets* (Set of 8 bit).
    - When it is written in *decimal number* each *Octet* ranges from *0 to 255.*
  - **IPv6** *(Internet Protocol version 6) :*
    - It is like a new thing. It has been out there for years.
    - Upgraded version of IPv4.
    - Many *ISPs* are implementing *IPv6* .
    - *128 bit hexa decimal number.*
- **There are 2 methods for assigning *IP Addresses :***

- **Dynamic :**
  when you go home you connect to *wifi router* that router has some enable services named *DHCP* which provides *IP Addresses* that your machine is getting. so you don't have to do anything in order to assign IP Address to a device.
- **Static:**
  we physically go to device and *statically assign IP Address* to servers/printers/other resources which provide services to end systems.

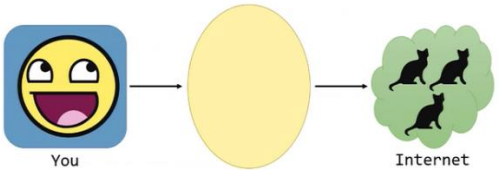
- **Classes Full and Class Less IP Addresses :**
- **5 Classes of IP Addresses (Class full IP Addresses) :**
  - More likely the ones which you need to know are 3 :
    - CLASS A - Reserved for Government Organisations.
    - CLASS B - Reserved for Medium Companies.
    - CLASS C - Reserved for Small Companies.
  - The other 2 are :
    - CLASS D - Reserved for Multicasting.
    - CLASS E - Reserved for Experimental Purposes.
- **Private & Public IP Addresses:**
  - **Private**
    IP Address are for *private networks.*It could be of 10 or 100 or may be thousands of thousands networks. Which will use same type of *IP Addresses* and that is completely fine because it's a private network.
  - **Public**
    IP Addresses of Web Servers. when we reach out a websites like Google, Yahoo , YouTube , Microsoft , Fb, they have hundreds of ip address to distribute all the traffic coming in. So it has to be unique.
- **IP Addresses are Unique:**
  IP Address of a system in a *local area network* should be *unique*. If you have 100 devices on a same network better make sure they do not have same *IP Address.*
  If we go to the *public area network* same again we have to make sure that the *IP Address* that a web server has no other web server or any other *network device* in *public network* should have.
- **IP Addresses Identifies a Device on a Network:**
  Yess.. it makes sense too. Once you have a ***subnet mask*** you will be able to identify a ***network Id*** *and* ***Host Id*** in an *IP Address.*
- **Network Id and Host Id :**
  An *IP Address* has *Network Id and Host Id*. *Network Id/Host Id* could be of any number of bits depending on the *class* and *subnet mask.*
- **Layer 3 of the OSI Model:**
  Network Layer ..yesss. This is where *IP Address* are. It is one of the *biggest protoco*l out there. **Internet Protocol** *(UDP protocol)*
- **Router**
  Routers reads *IP Addresses*. They are *Layer 3 Devices.* There is ***Routing Table*** which tells how to get from one network to another.
  Network A to Network B
  Network A to Network C
  Network B to Network C
  etc. Based on IP Addresses.
- **Packets:**
  We get *data* in form of **PDU** from top 3 layers and then when *data* enters in *4th layer i.e. Transport Layer* it becomes ***Segment*** and when that same segment enters in *Layer 3 i.e. Network Layer* it becomes a ***Packet.*** once you pull that packet you will see a ***Destination IP Address*** and a ***Source IP Address***
- **Routing Table:**
  Once you write a command '***show ip route***' you will see a lot depending upon how big your *network* is. You could see probably a hundreds of *IP Addresses.*
- ***DHCP (Pool of IP Addresses)***
  when we want *dynamic IP Address* allocation. We need *DHCP*. *Servers* like ***windows server 2012/ windows server 2016/windows server 2019, infovlox*** are used to *manager and configure DHCP*. Once *DHCP* is enabled as a role to one of the server then you can *assign pool of IP Address saying*
  for abc VLANS i want 123 IP Addresses.
  for xyz VLANS i want 456 IP Addresses.
  etc.
  DHCP provides
  Subnet Mask
  Default Gateway
  DNS
  and of course ***IP Address*** *to a device.*
  DHCP can also be set upped in a *Router.*
- **NIC Card :**
  Nic card are also assigned an *IP Address* to their ports.
- **Switches and Routers:**
  Different ports of *Switches* and *Routers* can be assigned different *IP Address.*

- **TCP/IP Settings:**
  Go to ***cmd*** and type **ipconfig/all** all *TCP/IP settings* will come up.
  DHCP provides up *TCP/IP* settings
- **ARP (Address Resolution Protocol):**
  MAC Addresses with same *IP Addresses*. In a *LAN, a* Switch connects different devices. You send a message from device A to device B and you only know the IP Address of device B so now you request a *switch* giving it the *IP Address* of *system B* to connect it but wait a minute *switch* works on *Physical addresses*. *Switches* won't get your request. So to tackle such situations/Problems ***ARP*** helps switch and provides MAC Address of device B.

Internet Primer

- We've reached the point in the course where we begin the transition away from the command-line oriented world of C and start considering the web-based world of PHP and the like.
- Before we dive headlong into that world, it's a good idea to have a basic understanding of how humans and computers interact over the internet.

Internet Primer



Internet Primer

- **IP Address**
- In order for your machine to uniquely identify itself on the Internet, it needs an address.
  - This way, it can send information out and also receive information back to the correct location.
- The addressing scheme used by computers is known as IP addressing.

Internet Primer

- **IP Address**
- As originally developed, the IP addressing scheme would effectively allocate a unique 32-bit address to each device hoping to connect to the internet.
- Instead of representing these 32-bit addresses as hexadecimal, we represent them as four clusters of 8-bits using decimal notation.

Internet Primer

- **IP Address**

`w.x.y.z`

- Each of w, x, y, and z can be a nonnegative value in the range [0, 255].

Internet Primer

- **IP Address**

`123.45.67.89`

- Each of w, x, y, and z can be a nonnegative value in the range [0, 255].

Internet Primer

- **IP Address**
- If each IP address is 32 bits, that means there are roughly 4 billion addresses to give out.
- The population of the world is somewhere in excess of 7 billion, and most folks in the western world have more than 1 device capable of Internet connectivity.
  - Some workarounds have allowed us to deal with this problem (for now).

Internet Primer

- **IP Address**
- In recent years, we've been slowly phasing out this old scheme (IPv4) and replacing it with a newer scheme (IPv6) that assigns computers 128-bit addresses, instead of 32-bit addresses.

`340,282,366,920,938,463,463,374,607,431,768,211,456`

Internet Primer

- **IPv6 Address**

`s:t:u:v:w:x:y:z`

- Each of s, t, u, v, w, x, y, and z is represented by 1 to 4 hexadecimal digits in the range [`0,ffff`].

Internet Primer

- **IPv6 Address**

`1234:5678:90ab:cdef:fedc:ba09:8765:4321`

- Each of s, t, u, v, w, x, y, and z is represented by 1 to 4 hexadecimal digits in the range [`0,ffff`].

Internet Primer

- **IPv6 Address**

`2001:4860:4860:0:0:0:0:8844`

- Each of s, t, u, v, w, x, y, and z is represented by 1 to 4 hexadecimal digits in the range [`0,ffff`].

Internet Primer

- **IPv6 Address**

`2001:4860:4860::8844`

- Each of s, t, u, v, w, x, y, and z is represented by 1 to 4 hexadecimal digits in the range [`0,ffff`].

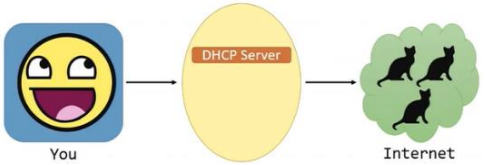
Internet Primer

- **DHCP**
- How do we get an IP address in the first place though? Surely we can't just choose any one we want. What if the want we want is already taken?
- Somewhere between your computer and the Internet at large exists a *Dynamic Host Configuration Protocol* (DHCP) server, whose role is to assign IP addresses to devices.

Internet Primer

- **DHCP**
- Prior to the widespread promulgation of DHCP, the task of assigning IP addresses fell to a system administrator, who would need to manually assign such addresses.

Internet Primer



Internet Primer

- **DNS**
- Odds are, you've never actually tried to visit a website by typing its IP address into your browser.
- The Domain Name System (DNS) exists to help us translate IP addresses to more memorable names that are more human-comprehensible.
- In this way, DNS is somewhat like the yellow pages of the web.

Internet Primer

- **DNS**

Host	IPv4 Address
info.host1.net	0.0.0.0
info.host2.net	0.0.0.1
...	
io-in-f138.1e100.net	74.125.202.138
...	
info.hostn-1.net	255.255.255.254
info.hostn.net	255.255.255.255

Internet Primer

- **DNS**

Host	IPv6 Address
info.host1.net	0:0:0:0:0:0:0:0
info.host2.net	0:0:0:0:0:0:0:1
...	
google.ie	2a00:1450:4010:c09:0:0:0:93
...	
info.hostn-1.net	ffff:ffff:ffff:ffff:ffff:ffff:ffff:fffe
info.hostn.net	ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff

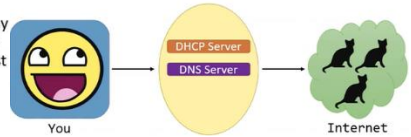
Internet Primer

- **DNS**
- Much like there is no yellow pages of the world, there is really no DNS record of the entire internet.
- Rather, large DNS server systems (like Google's own) are more like aggregators, collecting smaller sets of DNS information and pooling them together, updating frequently.

Internet Primer

- **DNS**
- In that way, large DNS servers are like libraries that stock many different sets of local yellow page books. In order to have the most up-to-date phone numbers for businesses, libraries must update the books they have on hand.
- DNS record sets are thus fairly decentralized.

Internet Primer



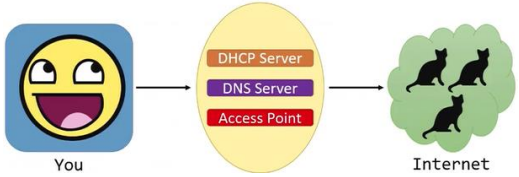
Internet Primer

- **Access Points**
- One of the ways we've dealt with the IPv4 addressing problem is to start assigning multiple people to the same IP address.
- The IP address is assigned to a *router*, whose job it is to act as a traffic cop that allows data requests from all of the devices on your local network (your home or business, e.g.) to be processed through a single IP address.

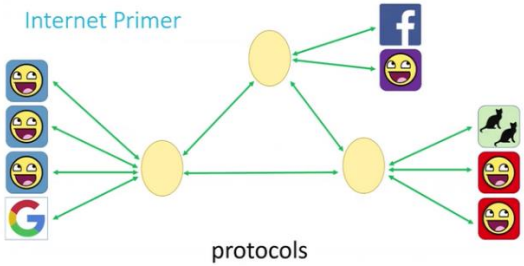
Internet Primer

- **Access Points**
- Modern home networks consist of access points that combine a router, a modem, a switch, and other technologies together into a single device.
- Modern business networks or large-scale wide-area networks (WANs) still frequently have these as separate devices to allow the size of their network to scale more easily.

Internet Primer



Internet Primer



Internet Protocol (IP)

- As discussed previously, "the Internet" is really an *interconnected network* comprised of smaller networks woven together and agreeing to communicate with one another.
- How do these networks know how to communicate with one another? This is the responsibility of the Internet Protocol (IP).
- Though it's admittedly on an extremely small scale, this picture is misleading as it pertains to network communication.

Internet Protocol (IP)

- With only six networks, things are rapidly getting out of hand. And the modern Internet consists of a lot more than six networks. We simply can't afford to wire them together such that each directly connects with every other.
- But still, we need each network to be able to talk to each other network, or we end up with pieces of the network that are unable to speak to other parts of the network.

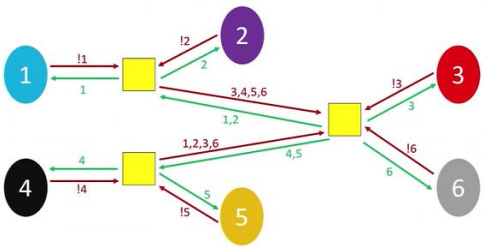
Internet Protocol (IP)

- This is where **routers** come back into play.
- What if, instead of being connected to every other network, each network was connected to a limited number of routers (each of which was connected to other nearby routers), and each router had instructions built into it on how to move information toward its destination?
- This information might be stored in a routing table, inside of the router.

Internet Protocol (IP)

- For this illustration, let's assume each network has IP addresses in the range of *n.x.x.x*, where *n* is its network number, and each x is in the range [0, 255].
  - A generalization of the way things actually work!

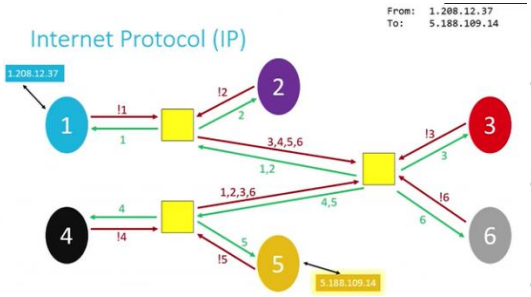
Internet Protocol (IP)



Internet Protocol (IP)

- Now the networks are not directly connected to each other at all, and rely on routers to distribute communications.
- On a small scale, this configuration may actually be more inefficient than just having direct connections.
- On a large scale, this configuration can dramatically reduce the costs of network infrastructure.

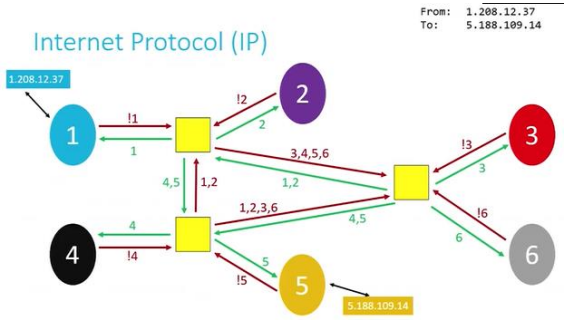
Internet Protocol (IP)



Internet Protocol (IP)

- In reality, if 1.208.12.37 (me) is sending an e-mail, FTP file transfer, or web browser request to 5.188.109.14 (you), the data isn't being sent as one huge block.
- Any slowdown that was caused by sending such a large amount of data would have a ripple effect that would throttle the network for all the other users.
- As such, another crucial part of IP is splitting data into *packets*.

Internet Protocol (IP)



Internet Protocol (IP)

- IP is also known as a *connectionless* protocol. There is not necessarily a defined path from the sender to the receiver, and vice versa.
- This means that in response to traffic that might be "clogging" up one particular path through the Internet, some packets can be "re-routed" around the traffic jam to follow the most optimal path, based on the current state of the network.

Internet Protocol (IP)

- The routing table then probably consists of more information than just "where do I send this packet from here," but also information "what is the cost of using that particular route?"
- Another side of effect of being connectionless is that delivery cannot be guaranteed, since the path from sender to receiver is not guaranteed to be consistent.

Transmission Control Protocol (TCP)

- If the Internet Protocol (IP) is thought of as the protocol for getting information from a sending machine to a receiving machine, then Transmission Control Protocol (TCP) can be thought of as directing the transmitted packet to the correct program on the receiving machine.
- As you might imagine, it is important to be able to identify both *where* the receiver is and *what* the packet is for, so TCP and IP are almost an inseparable pair: TCP/IP.

Transmission Control Protocol (TCP)

- Each program/utility/service on a machine is assigned a *port number*. Coupled with an IP address, we can now uniquely identify a specific program on a specific machine.
- The other thing that TCP is crucial for is *guaranteeing delivery* of packets, which IP alone does not do.
- TCP does this by including information about how many packets the receiver should expect to get, and in what order, and transmitting that information alongside the data.

Transmission Control Protocol (TCP)

- Some ports are so commonly used that they have been standardized across all computers.
  - FTP (file transfer) uses port **21**.
  - SMTP (e-mail) uses port **25**.
  - DNS uses port **53**.
  - HTTP (web browsing) uses port **80**.
  - HTTPS (secure web browsing) uses post **443**.

Transmission Control Protocol (TCP)

- Steps of the TCP/IP process
  1. When a program goes to send data, TCP breaks it into smaller chunks and communicates those packets to the computer's network software, adding a TCP layer onto the packet.
  2. IP routes the individual packets from sender to receiver; this info is part of the IP layer surrounding the packet.
  3. When the destination computer gets the packet, TCP looks at the header to see which program it belongs to; and since the routes packets take may differ, TCP also must present those packets to the destination program in the proper order.

Transmission Control Protocol (TCP)

SENDER

RECEIVER

Transmission Control Protocol (TCP)

SENDER

RECEIVER

Transmission Control Protocol (TCP)

SENDER

RECEIVER

Transmission Control Protocol (TCP)

- If at any point along the way a router delivering information using the Internet Protocol *dropped* a packet, TCP would use additional information inside the headers to request that the sender pass along the extra packet so it could complete assembly.
- After the packets have arrived, TCP ensures they are organized the correct order and can then be reassembled into the intended unit of data and delivered to the correct service.

Running a Traceroute

Take the following steps to run a traceroute in Microsoft® Windows®:

1. Press **Windows key** + **R** to open the **Run** window.
2. Enter **cmd** and press **Enter** to open a **Command Prompt**.
3. Enter **tracert**, a space, then the IP address or web address for the destination site (for example: tracert www.lexis.com).
4. Press **Enter**.

Lookup A record:

```
nslookup -type=A hostens.com
```

Example output:

```
C:\Windows\system32>nslookup -type=A hostens.com
Server:  rs8-lt1.serveriai.lt
Address:  212.24.109.143

Non-authoritative answer:
Name:    hostens.com
Addresses:  104.20.2.91
          104.20.3.91
```

If you are using windows, open CMD and type "ping www.xyz.com" this will provide you with the servers IP address .

Hypertext Transfer Protocol (HTTP)

- In addition to protocols that dictate how information is communicated from machine to machine and application to application (IP and TCP, respectively), it is frequently the case that the application itself has a system of rules for how to interpret the data that was sent.
- HTTP is one such example of an *application layer protocol*, which specifically dictates the format by which clients request web pages from a server, and the format via which servers return information to clients.

Hypertext Transfer Protocol (HTTP)

- Other application layer protocols include:
  - File Transfer Protocol (FTP)
  - Simple Mail Transfer Protocol (SMTP)
  - Data Distribution Service (DDS)
  - Remote Desktop Protocol (RDP)
  - Extensible Message and Presence Protocol (XMPP)

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

- A line of the form

```
method request-target http-version
```

- Is a simple example of an *HTTP request line*, a crucial part of an overall HTTP request that a client may make to a server.

Hypertext Transfer Protocol (HTTP)

- A line of the form

```
GET request-target HTTP/1.1
```

- Is a simple example of an *HTTP request line*, a crucial part of an overall HTTP request that a client may make to a server.

Hypertext Transfer Protocol (HTTP)

- A line of the form

```
POST request-target HTTP/1.1
```

- Is a simple example of an *HTTP request line*, a crucial part of an overall HTTP request that a client may make to a server.

Hypertext Transfer Protocol (HTTP)

- The host name (domain name of the server) is also included as a separate line of the overall HTTP request.
- Taken together, the host name and the request target from the request line specify a specific resource being sought.
- Based on whether the resource exists and whether the server is empowered to deliver that resource pursuant to the client's request, a number of *status codes* can result.

Hypertext Transfer Protocol (HTTP)

Class	Code	Text	Comments
Success	200	OK	All is well, valid request and response.
Redirection	301	Moved Permanently	Page is now at a new location, automatic redirects built in to most browsers.
	302	Found	Page is now at a new location *temporarily*.
Client Error	401	Unauthorized	Page typically requires login credentials.
	403	Forbidden	Server will not allow this request.
	404	Not Found	Server cannot find what was asked for.
Server Error	500	Internal Server Error	Generic server failure in responding to the otherwise-valid request.

HTML

- HTML (*Hypertext Markup Language*) is a fundamental component of every website.
- HTML is a language, but it is not a programming language. It lacks concepts of variables, logic, functions, and the like.
- Rather, it is a *markup language*, using angle-bracket enclosed tags to semantically define the structure of a web page, causing the plain text inside of sets of tags to be interpreted by web browsers in different ways.

HTML

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

HTML

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

HTML

- Notice how the markup allows us to convey extra information about the text we've written.
- There are over 100 HTML tags, and lots of great resources online to find them. We won't cover them all here.
- Another interesting way to learn about HTML tags is to view the source of a website you frequent by opening up your browser of choice's developer tools.

HTML

- **Common HTML tags**
  - `<b>, </b>`
    - Text between these tags will be rendered in **boldface** by the browser.
  - `<i>, </i>`
    - Text between these tags will be rendered in *italics* by the browser.
  - `<u>, </u>`
    - Text between these tags will be rendered underlined by the browser.

/cs50/sections/week08/biu.html

This text is **bold**, this text is *italic*, and this text is underlined.

HTML

- **Common HTML tags**
  - `<p>, </p>`
    - Text between these tags will be rendered as a paragraph by the browser, with space above and below.
  - `<hX>, </hX>`
    - X = 1, 2, 3, 4, 5, or 6
    - Text between these tags will be rendered as an X-level section header.

HTML

- **Common HTML tags**
  - `<ul>, </ul>`
    - Demarcate the beginning and end of an unordered (bulleted) list.
  - `<ol>, </ol>`
    - Demarcate the beginning and end of an ordered (numbered) list.
  - `<li>, </li>`
    - Demarcate list items with an ordered or unordered list.

- alpha
- beta
- gamma
- delta
- epsilon

6. zeta
7. eta
8. theta
9. iota
10. kappa

```
<html>
    <body>
        <ul>
            <li>alpha</li>
            <li>beta</li>
            <li>gamma</li>
            <li>delta</li>
            <li>epsilon</li>
        </ul>
        <ol start=6>
            <li>zeta</li>
            <li>eta</li>
            <li>theta</li>
            <li>iota</li>
            <li>kappa</li>
        </ol>
    </body>
</html>
```

HTML

- **Common HTML tags**
  - `<table>, </table>`
    - Demarcate the beginning and end of a table definition.
  - `<tr>, </tr>`
    - Demarcate the beginning and end of a row within a table.
  - `<td>, </td>`
    - Demarcate the beginning and end of a column within a row within a table.

HTML

- **Common HTML tags**
  - `<form>, </form>`
    - Demarcate the beginning and end of an HTML form.
  - `<div>, </div>`
    - Demarcate the beginning and end of an arbitrary HTML page division.
  - `<input name=X type=Y />`
    - Define a field within an HTML form. X is a unique identifier for that field, Y is what type of data it accepts.

```
<html>
    <body>
        <div>
            Arbitrary first division
        </div>

        Here is a form:
        <div>
            <form>
                Text: <input name="a" type="text" />
                Password: <input name="b" type="password" />
                Radio: <input name="c" type="radio" />
                Checkbox: <input name="d" type="checkbox" />
                Submit: <input name="e" type="submit" />
            </form>
        </div>
    </body>
</html>
```

Arbitrary first division
Here is a form:
Text: Password: Radio: Checkbox: Submit: Submit

HTML

- **Common HTML tags**
  - `<a href=X>, </a>`
    - Creates a hyperlink to web page X, with the text between the tags rendered and functional as the link text.
  - `<img src=X ... />`
    - Another self-closing tag for displaying an image located at X, with possible additional *attributes* (such as specifying width and height).
  - `<!DOCTYPE html>`
    - Specific to HTML5, lets the browser know that's the standard you're using.

HTML

- **Common HTML tags**
  - `<!--, -->`
    - Demarcate the beginning and end of an HTML comment.

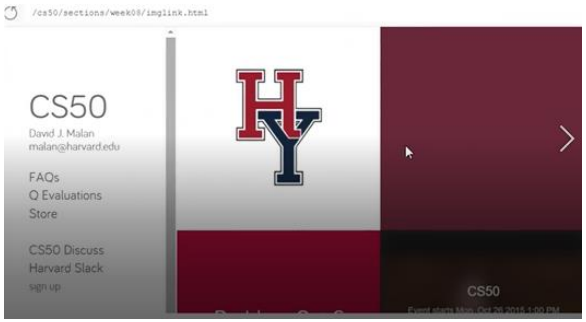
- Beyond the tags as explained here, each can also have multiple *attributes* that slightly modify the tag.
  - For example, you can usually add an `id`=X attribute, to uniquely identify a tag within an overall page.

```
<html>
    <body>
        <div>
            <!-- Here is a picture of Rick Astley -->
            <img src="rick.jpg" />
        </div>
        <div>
            <!-- Here is an internal link to another page -->
            <a href="hello.html">To hello.html</a>

            <!-- Here is an external link -->
            To <a href="https://cs50.harvard.edu">CS50's webs
        </div>
    </body>
</html>
```

To hello.html To CS50's website

/cs50/sections/week08/imglink.html

World, hello



HTML

- It is important that the HTML you write be well-formed. Every tag you open should be closed (unless it is a self-closing tag), and tags should be closed in reverse order of when they were opened.
- Unlike C, your HTML will not necessarily fail with syntax errors if not well-formed, so it's up to you to be vigilant.
- Because it can be an arduous task to investigate this, be sure to use online HTML validators to help!
  - **You can use validator.w3.org -> check to see if on other sites the page will appear ok**

CSS

- CSS (*Cascading Style Sheets*) is another language we use to when constructing websites.
  - If HTML is used to organize the content that we aim to display on our pages, then CSS is the tool we use to customize our website's look and feel.
- Like HTML, CSS is not a programming language; it lacks logic. Rather, it is a styling language and its syntax describes how certain attributes of HTML elements should be modified.

CSS

```
body
{
    background-color: blue;
}
```

CSS

- A style sheet is constructed by identifying a *selector* (in the last example, body) and then an open curly brace to indicate the beginning of the style sheet for that selector.
- In between the curly brace you place a list of key-value pairs of style properties and values for those properties, each *declaration* ending with a semicolon.
- Then a closing curly brace terminates the style sheet.

CSS

- **Common CSS properties**
  - `border: style color width`
    - Applies a border of the specified color, width, and style (e.g., dotted, dashed, solid, ridge…).
  - `background-color: [keyword | #<6-digit hex>]`
    - Sets the background color. Some colors are pre-defined in CSS.
  - `color: [keyword | #<6-digit hex>]`
    - Sets the foreground color (usually text).

CSS

- **Common CSS properties**
  - `font-size: [absolute size | relative size]`
    - Can use keywords (xx-small, medium…), fixed points (10pt, 12pt…), percentage (80%, 120%), or base off the most recent font size (smaller, larger).
  - `font-family: [font name | generic name]`
    - Certain "web safe" fonts are pre-defined in CSS.
  - `text-align: [left | right | center | justify]`
    - For displaying text.

CSS

- Your selectors don't have to apply only to HTML tag categories. There also exist ID selectors and class selectors.
- A **tag** selector will apply to all elements with a given HTML tag.

```
h2
{
    font-family: times;
    color: #fefefe;
}
```

CSS

- Your selectors don't have to apply only to HTML tag categories. There also exist ID selectors and class selectors.
- An **ID** selector will apply only to an HTML tag with a unique identifier.

```
#unique
{
    border: 4px dotted blue;
    text-align: right;
}
```

CSS

- Your selectors don't have to apply only to HTML tag categories. There also exist ID selectors and class selectors.
- A **class** selector will apply only to those HTML tags that have been given identical "class" attributes.

```
.students
{
    background-color: yellow;
    opacity: 0.7;
}
```

CSS

- Style sheets can be written directly into your HTML.
  - Place them within `<style>` tags within your page's head.
- Style sheets can also be written as separate CSS files and then linked in to your document.
  - Use `<link>` tags within your page's head to accomplish this.

```
<html>
    <head>
        <style>
            table
            {
                border: 1px solid blue;
            }
        </style>
    </head>
```

```
<html>
    <head>
        <link href="tables.css" rel="stylesheet" />
    </head>
```

```
table
{
    border: 5px solid red;
}

tr
{
    height: 50px;
}

td
{
    /* Lots being applied here! */

    background-color: black;
    color: white;
    font-size: 22pt;
}
```

/cs50/sections/week08/table3.html

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

JavaScript

- JavaScript is a modern programming language that is derived from the syntax at C.
- It has been around just about as long as Python, having been invented a few years later, in 1995.
- JavaScript, HTML, and CSS make up the three languages defining most of the user experience on the web.

JavaScript

- To start writing JavaScript, open up a file with the .js file extension.
- No need for any code delimiters like you may be familiar with if you've used a language like PHP. Our website will know that our file is JavaScript because we'll explicitly tell it as much in an HTML tag.
- Unlike Python which runs *server-side*, JavaScript applications run *client-side*, on your own machine.

JavaScript

- **Including JavaScript in your HTML**
- Just like CSS with `<style>` tags, you can directly write your JavaScript between `<script>` tags.
- Just like CSS with `<link>` tags, you can write your JavaScript in separate files and link them in by using the src attribute of the `<script>` tag.

JavaScript

- **Variables**
- JavaScript variables are similar to Python variables.
  - No type specifier.
  - When a local variable is first declared, preface with the `var` keyword.

```
var x = 44;
```

JavaScript

- **Conditionals**
- All of the old favorites from C are still available for you to use.

```
if
else if
else
switch
?:
```

JavaScript

- **Loops**
- All of the old favorites from C are still available for you to use.

```
while
do-while
for
```

JavaScript

- **Functions**
- All functions are introduced with the `function` keyword.
- JavaScript functions, particularly those bound specifically to HTML elements, can be *anonymous*—you don't have to give them a name!
  - We'll revisit anonymity a little later, and we'll revisit "binding to HTML elements" in the video on the Document Object Model.

JavaScript

- **Arrays**
- Declaring an array is pretty straightforward.

```
var nums = [1, 2, 3, 4, 5];
var mixed = [1,
             true,
             3.333,
             'five'];
```

s

JavaScript

- **Objects**
- JavaScript has the ability to behave in a few different ways, but in particular it can behave as an *object-oriented* programming language.
- An object is sort of analogous to a C structure.

JavaScript

- **Objects**
- C structures contain a number of *fields* or *members*, which we might also call *properties*.
  - But the properties themselves can not ever stand on their own.

```
struct car
{
   int year;
   char model[10];
}
```

```
struct car herbie;
herbie.year = 1963;
herbie.model = "Beetle";
```

JavaScript

- **Objects**
- C structures contain a number of *fields* or *members*, which we might also call *properties*.
  - But the properties themselves can not ever stand on their own.

```
struct car
{
   int year;
   char model[10];
}
```

```
struct car herbie;
year = 1963;
model = "Beetle";
```

JavaScript

- **Objects**
- C structures contain a number of *fields* or *members*, which we might also call *properties*.
  - But the properties themselves can not ever stand on their own.
- Objects, meanwhile, have properties but also *methods*, or functions that are inherent to the object, and mean nothing outside of it.
  - Thus, like properties can methods not ever stand on their own.

JavaScript

- **Objects**

```
object.function();
```

JavaScript

- **Objects**

- The fields and methods of an object are similar in spirit to the idea of a dictionary, with which we're familiar from Python.

```
var herbie = {year : 1963, model: 'Beetle'};
```

JavaScript

- **Loops (redux)**

- How do we iterate across all of the key-value pairs of an object (or indeed, all of the elements of an array)?

```
for (var key in object)
{
   // use object[key] in here
}
```

JavaScript

- **Loops (redux)**

- How do we iterate across all of the key-value pairs of an object (or indeed, all of the elements of an array)?

```
for (var key of object)
{
   // use key in here
}
```

JavaScript

- **Loops (redux)**

```
var wkArray = ['Monday',
               'Tuesday',
               'Wednesday'
               'Thursday',
               'Friday',
               'Saturday',
               'Sunday'];

for (var day in wkArray)
{
   console.log(day);
}
```

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

JavaScript

- **Loops (redux)**

```
var wkArray = ['Monday',
               'Tuesday',
               'Wednesday',
               'Thursday',
               'Friday',
               'Saturday',
               'Sunday'];

for (var day of wkArray)
{
   console.log(day);
}
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

JavaScript

- **Printing and variable interpolation**

```
console.log(wkArray[day] + ' is day number '
            + (parseInt(day) + 1) +
            ' of the week!');
```

JavaScript

- **Functions (redux)**

- Arrays are a special case of an object (in fact, *everything* in JavaScript is a special case of an object), and has numerous methods that can applied to them:
  - `array.size(), array.pop(), array.push(x), array.shift();`

- There is also a method for arrays called `map()`, which can be used to apply a function to all elements of an array.
  - A great situation to use an *anonymous function*

JavaScript

- **Functions (redux)**

```
var nums = [1, 2, 3, 4, 5];

nums = nums.map(function(num) {
   return num * 2;
});
```

JavaScript

- **Events**

- An *event* in HTML and JavaScript is a response to user interaction with the web page.
  - A user clicks a button, a page has finished loading, a user has hovered over a portion of the page, the user typed in an input field.

- JavaScript has support for *event handlers*, which are callback functions that respond to HTML events.
  - Many HTML elements have support for events as an attribute.

JavaScript

```
<html>
    <head>
        <title>Event Handlers</title>
    </head>
    <body>
        <button onclick="">Button 1</button>
        <button onclick="">Button 2</button>
    </body>
</html>
```

JavaScript

- **Events**

- We can write a generic event handler in JavaScript, creating an *event object*, that will tell us which of these two buttons was clicked.

JavaScript

```
<html>
    <head>
        <title>Event Handlers</title>
    </head>
    <body>
        <button onclick="alertName(event)">Button 1</button>
        <button onclick="alertName(event)">Button 2</button>
    </body>
</html>
```

JavaScript

```
function alertName(event)
{
    var trigger = event.srcElement;
    alert('You clicked on ' + trigger.innerHTML);
}
```

PHP Syntax

- PHP is a great example of a commonly-used modern programming language.
  - C was first released in 1972, PHP in 1995.
- PHP is an excellent language choice for software that requires an easy way to do things that, in C, are really complicated.
  - String manipulation
  - Networking
- Fortunately, PHP is heavily inspired by C (in fact, the PHP interpreter is written in C) and so the syntax should be pretty familiar.

PHP Syntax

- To start writing PHP, open up a file with the .php file extension.
- All PHP code inside of that file must be enclosed in the PHP delimiters.

```
<?php

?>
```

- If not, when the code is run, everything outside of those delimiters will simply be printed out verbatim in your program.

PHP Syntax

- Unlike C, which is a **compiled** language, PHP is an **interpreted** language.
  - Get to skip the step of compiling our code, with the trade off that PHP code needs to be converted to 0s and 1s on the fly, which might slow the program down.
- The fact that the PHP interpreter effectively "discards" non-PHP can actually be used to our advantage though, particularly when we start to mix PHP and HTML in the context of web programming.

PHP Syntax

- **Variables**
- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

```
$phrase = "This is CS50";
```

PHP Syntax

- **Conditionals**
- All of the old favorites from C are still available for you to use.

```
if ($coursenum == 50)
{

}
else if ($name == "David")
{

}
```

PHP Syntax

- **Conditionals**
- All of the old favorites from C are still available for you to use.

```
if ($coursenum == 50)
{

}
elseif ($name == "David")
{

}
```

PHP Syntax

- **Conditionals**
- All of the old favorites from C are still available for you to use.

```
$var = 'A';
$letter = ctype_alpha($var) ? true : false;
```

PHP Syntax

- **Conditionals**
- All of the old favorites from C are still available for you to use.

```
$state = readline("Your state, please: ");
switch($state)
{
   case "MA": case "Mass.": case "Massachusetts":
      // stuff
      break;
   default:
      echo("I don't understand your input!");
}
```

PHP Syntax

- **Loops**
- All of the old favorites from C are still available for you to use.

```
$counter = 1;
while($counter <= 100)
{
   print($counter);
   $counter++;
}
```

PHP Syntax

- **Loops**
- All of the old favorites from C are still available for you to use.

```
$counter = 1;
do
{
   print($counter);
   $counter++;
}
while($counter <= 100);
```

PHP Syntax

- **Loops**
- All of the old favorites from C are still available for you to use.

```
for ($counter = 1; $counter <= 100; $counter++)
   print($counter);
```

PHP Syntax

- **Arrays**
- Here's where things really start to get a lot better than C.
- PHP arrays are **not** fixed in size; they can grow or shrink as needed, and you can always tack extra elements onto your array and splice things in and out easily.
- Because PHP is loosely typed, your arrays can also mix different data types together.

PHP Syntax

- **Arrays**
- Declaring an array is pretty straightforward… though there is more than one way to do it.

```
$nums = [1, 2, 3, 4];
```

PHP Syntax

- **Arrays**
- Recall that our arrays also do not have to always contain elements of the same data type.

```
$nums = [1, true, 3, 4];
array_push($nums, "five");
```

PHP Syntax

- **Arrays**
- Tacking on to an existing array can be done a few ways:

```
$nums = [1, 2, 3, 4];
$nums[4] = 5;
```

PHP Syntax

- **Associative arrays**
- PHP also has built in support for **associative arrays**, allowing you to specify array indices with words or phrases (*keys*), instead of integers, which you were restricted to in C.

PHP Syntax

- **Associative arrays**

```
$pizzas = [
    "cheese" => 8.99,
    "pepperoni" => 9.99,
    "vegetable" => 10.99,
    "buffalo chicken" => 11.99
];
```

PHP Syntax

- **Associative arrays**

```
$pizza["cheese"] = 7.99;

if ($pizza["vegetable"] < 12)
{

}

$pizza["bacon"] = 13.49;
```

PHP Syntax

- **Associative arrays**
- PHP also has built in support for **associative arrays**, allowing you to specify array indices with words or phrases (*keys*), instead of integers, which you were restricted to in C.
- But this creates a somewhat new problem… how do we iterate through an associative array? We don't have indexes ranging from [0, *n*-1] anymore.

PHP Syntax

- **Loops (redux)**
- PHP also includes a new kind of loop called **foreach** that allows you to iterate across any array, but is particularly useful for associative arrays.

```
foreach($array as $key)
{
   // use $key in here as a stand-in for $array[$i]
}
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    "cheese" => 8.99,
    "pepperoni" => 9.99,
    "vegetable" => 10.99,
    "buffalo chicken" => 11.99
];

foreach($pizzas as $pizza)
{
   print($pizza);
   print("\n");
}
```

```
8.99
9.99
10.99
11.99
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    "cheese" => 8.99,
    "pepperoni" => 9.99,
    "vegetable" => 10.99,
    "buffalo chicken" => 11.99
];

foreach($pizzas as $topping => $price)
{
   print("A whole ");
   print($topping);
   print(" pizza costs $");
   print($price);
   print(".\n");
}
```

```
A whole cheese pizza costs $8.99.
A whole pepperoni pizza costs $9.99.
A whole vegetable pizza costs $10.99.
A whole buffalo chicken pizza costs $11.99.
```

PHP Syntax

- **Printing and variable interpolation**
- We've already seen a few ways to print out information to the user.
  - `print()`
  - `echo()`
- Suffice it to say, PHP has a lot of ways to print, including ways to print out substituted variables in the middle of strings.

PHP Syntax

- **Printing and variable interpolation**

```
print("A whole ");
print($topping);
print(" pizza costs $");
print($price);
print(".\n");
```

PHP Syntax

- **Printing and variable interpolation**

```
printf("A whole %s pizza costs $%s.\n",
       $topping, $price);
```

PHP Syntax

- **Printing and variable interpolation**

```
echo("A whole {$topping} pizza costs
      \${$price}.\n");
```

PHP Syntax

- **Printing and variable interpolation**

```
print("A whole " . $topping .
      " pizza costs $" . $price . ".\n");
```

PHP Syntax

- **Functions**
- PHP has support for functions as well. Like variables, we don't need to specify the return type of the function (because it doesn't matter), nor the data types of any parameters (ditto).
- All functions are introduced with the `function` keyword.
  - Also, no need for `main()`, since the interpreter just reads from top to bottom!

PHP Syntax

- **Functions**

```
function hard_square($x)
{
   $result = 0;
   for ($i = 0; $i < $x; $i++)
      $result += $x;
   return $result;
}
```

PHP Syntax

```
<?php
   $x = readline("Give me a number to square: ");
   echo(hard_square($x) . "\n");

   function hard_square($x)
   {
      $result = 0;
      for ($i = 0; $i < $x; $i++)
         $result += $x;
      return $result;
   }
?>
```

PHP Syntax

- **Functions**

```
function hard_square($x = 10)
{
   $result = 0;
   for ($i = 0; $i < $x; $i++)
      $result += $x;
   return $result;
}
```

PHP Syntax

```
<?php
   echo(hard_square() . "\n");

   function hard_square($x = 10)
   {
      $result = 0;
      for ($i = 0; $i < $x; $i++)
         $result += $x;
      return $result;
   }
?>
```

PHP Syntax

- Though PHP is primarily used (and was initially designed) as a language for web-based programming, it is a full language and can do nearly all things C can, which means it can be run from the command line.
- All that is required is that the PHP interpreter is installed on the system you wish to run your PHP programs on.

PHP Syntax

- **Including files**
- Just like C programs can consist of multiple files to form a single program, so can PHP programs tie files together.

```
require_once(__DIR__ . "cs50.php");
```

PHP Syntax

- To run your PHP program through the PHP interpreter at the command-line, simply type

```
php <file>
```

- and your program will run through the interpreter, which will execute everything inside of the PHP delimiters, and simply print out the rest.

```
1 <?php
2    echo("hello, world\n");
3 ?>
```

```
Terminal
lloyd1@ide50:~/workspace/cs50/sections/week09/php $ php hello1.php
hello, world
lloyd1@ide50:~/workspace/cs50/sections/week09/php $
```

```
lloyd1@ide50:~/workspace/cs50/sections/week09/php $ php hello2.php
What is your name?
Doug
hello, Doug
lloyd1@ide50:~/workspace/cs50/sections/week09/php $
```

```
1 <?php
2    $name = readline("What is your name?\n");
3    print("hello, {$name}\n");
4 ?>
```

```
1 What is your name?
2 <?php
3    $name = readline();
4    print("hello, {$name}\n");
5 ?>
```

```
Terminal
lloyd1@ide50:~/workspace/cs50/sections/week09/php $ php hello3.php
What is your name?
Doug
hello, Doug
lloyd1@ide50:~/workspace/cs50/sections/week09/php $
```

PHP Syntax

- Recall that you don't have to have only one set of PHP delimiters in your program. You can have as many as you'd like, opening and closing them as needed.

```
<?php
    print("Please give me a number\n");
    $num1 = readline();
    print("Please give me a second number\n");
    $num2 = readline();
    $sum = $num1 + $num2;
    print("The sum of these two numbers is {$sum}\n");
?>
```

```
1  Please give me a number
2  <?php $num1 = readline() ?>
3  Please give me a second number
4  <?php $num2 = readline() ?>
5  The sum of these two numbers is <?= ($num1 + $num2) . "\n"?>
```

```
1  You rolled a <?= rand() % 6 + 1 ?> and a <?= rand() % 6 + 1 . "\n" ?>
```

PHP Syntax

- You can also make your programs look a lot more like C programs when they execute by adding a **shebang** to the top of your PHP files, which automatically finds and executes the interpreter for you.

```
#!/usr/bin/php
```

- If you do this, you need to change the **permissions** on your file as well using the linux command chmod as follows:

```
chmod a+x <file>
```

```
#!/usr/bin/php
Please give me a number
<?php $num1 = readline() ?>
Please give me a second number
<?php $num2 = readline() ?>
The sum of these two numbers is <?= ($num1 + $num2) . "\n"?>
```

Web Programming with PHP

- We know that we can use HTML to build websites, but websites built using pure HTML suffer from a serious limitation.
- Imagine we want to create a website that displays the current time in Cambridge, MA, displaying it to the latest minute.

Web Programming with PHP

```
<html>
   <head>
      <title>Current Time in Cambridge</title>
   </head>
   <body>
      The current time in Cambridge is 14:08
   </body>
</html>
```

Web Programming with PHP

- Websites that are pure HTML are completely static. The only way we can update the content of our pages is to manually open up our source files, edit and save, and then the next time the user visits or refreshes the page they'll get the content.
- Incorporating PHP into our code can make our code quite a bit more flexible and introduce a way for our pages to update or be dynamic without requiring our intervention.

Web Programming with PHP

- To run an instance of Apache (a popular web server with the capacity to interpret PHP), simply create a directory where you would like your pages to live and then run the command

```
apache50 start /path/to/dir
```

Terminal

```
lloyd1@ide50:~/workspace/cs50/sections/week09 $ ls
php/  php-web/
lloyd1@ide50:~/workspace/cs50/sections/week09 $ apache50 st
Setting Apache's document root to /home/ubuntu/workspace/cs
 * Starting web server apache2
 *
Apache started successfully!
Your site is now available at https://ide50-lloyd1.cs50.io
```

Web Programming with PHP

- To test that your Apache server is configured correctly, here's a common equivalent of "hello, world" for PHP web programming:

```
<?php phpinfo(); ?>
```

Web Programming with PHP

```
<?php
    date_default_timezone_set('US/Eastern');
    $time = date('H:i:s', time());
?>

<html>
   <head>
      <title>Current Time in Cambridge</title>
   </head>
   <body>
      The current time in Cambridge is <?= $time ?>.
   </body>
</html>
```

Web Programming with PHP

- Because the PHP interpreter ignores everything that is not inside of the PHP delimiters, that means that we can intersperse HTML and PHP without any problems, only using PHP for those parts of our website that require dynamism, and letting the static information remain the same.

- Why is it advantageous for us to do this? Instead of putting everything inside of the PHP delimiters and printing out HTML?

Web Programming with PHP

- There's not too much to writing PHP code for the web that's different from writing it for the command line... so far.

- What if we want to pass information **between** files that we have on our PHP server? Or what if we want to allow the user to pass information **into** our web pages?

- PHP supports a number of special *superglobal variables* to accomplish this.

Web Programming with PHP

- **$_GET**

- $_GET is perhaps the simplest of these superglobal variables for passing data between PHP files or allowing the user to supply your PHP page with data.

- With this technique, we scrape data supplied at the URL which is stored in a **query string** consisting of key-value pairs, which are then stored in the $_GET associative array.

Web Programming with PHP

```
<?php

    foreach($_GET as $key => $value)
    {
        print("<p>{$key}: {$value}</p>");
    }
?>
```

https://ide50-lloyd1.cs50.io/get1.php?name=Doug&year=2015&class=cs50

name : Doug

year : 2015

class : cs50

```
<!DOCTYPE html>

<html>
    <head>
        <title>Contents of Get (v.2)</title>
        <link href="get.css" rel="stylesheet" />
    </head>
    <body>
        <table>

        <!-- fancy new way to do a PHP foreach -->
        <?php foreach($_GET as $key => $value): ?>
            <tr>
                <td class="key"><?= $key ?></td>
                <td class="value"><?= $value ?></td>
            </tr>
        <?php endforeach; ?>

        </table>
```

https://ide50-lloyd1.cs50.io/get2.php?name=Doug&year=2015&class=cs50

name	Doug
year	2015
class	cs50

Web Programming with PHP

- **$_GET**
- Can you envision one possible issue with the $_GET superglobal?

http://localhost/get3.php?name=Doug&pw=squadgoals

Web Programming with PHP

- **$_GET**
- Can you envision one possible issue with the $_GET superglobal?
- $_GET is great for getting information to the page quickly, but can be problematic for passing *sensitive* information, as it makes it very easy for an attacker (or someone less malicious) to easily see that sensitive information.

Web Programming with PHP

- **$_POST**
- $_POST is an alternative method for transmitting data between pages or allowing users to send information that most frequently comes up in the context of HTML forms.
  - Though information can be sent via HTML forms with the GET method, too!
- Unlike $_GET, using this technique submits user information inside of the HTTP headers, instead of via the URL.

Web Programming with PHP

```
<html>
   <head>
      <title>Posting Form</title>
   </head>
   <body>
      <form action="post.php" method="post">
         <input name="name" type="text" />
         <input name="pw" type="password" />
         <input type="submit" />
      </form>
   </body>
</html>
```

```
1  <!DOCTYPE html>
2
3  <html>
4      <head>
5          <title>Posting Results</title>
6      </head>
7      <body>
8          <p>The user's name is <?= $_POST["name"] ?>.</p>
9          <p>The user's password is <?= $_POST["pw"] ?>.</p>
10     </body>
11 </html>
```

Web Programming with PHP

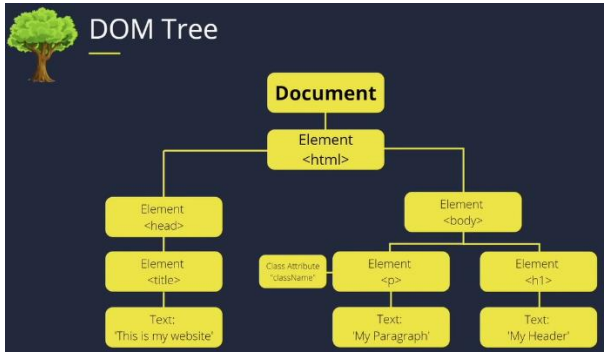
- **$_SESSION**
- $_SESSION is one additional superglobal that PHP provides to store data that you'd like to be available across multiple pages so long as a user is logged in to your website, so that you don't have to repeatedly pass information via $_POST.
- After you create a session with the PHP function `session_start()`, you can use and modify $_SESSION across your site and its contents will not be lost until you `session_destroy()`.

Web Programming with PHP

- Writing fairly dynamic websites with PHP drastically improves your site's quality without requiring too much additional complexity.
- Practice, practice, practice! You already know the basics from your experience with C, now you're just applying those basics with a slightly different language and in a slightly different context.
- Don't be afraid to look things up on the internet! `php.net`, for example, is a fantastic resource for PHP's documentation!

Document Object Model

- As we've seen, JavaScript objects are incredibly flexible, and can contain various fields, even when those fields are other objects.
- The *document object* is one way of employing this paradigm, whereby that object organizes the entire contents of a web page.
- By organizing an entire page into a JavaScript object, we can manipulate the page's elements programmatically.

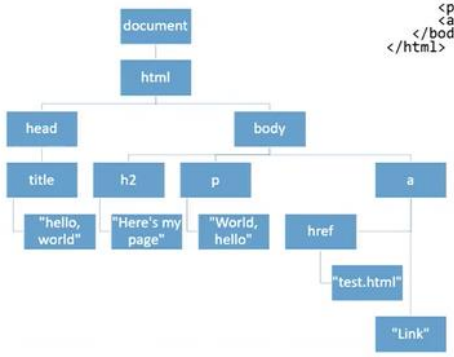


Document Object Model

```
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
      <a href="test.html">Link</a>
   </body>
</html>
```

• **Type in the console: console.dir(document)= this will organize the content of the page in a directory structure just like an object** •

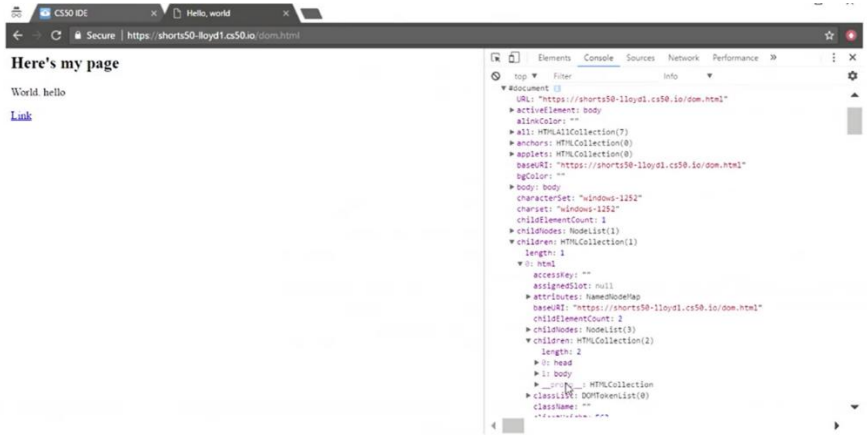
Document Object Model




```
<html>
   <head>
      <title>Hello, world</title>
   </head>
   <body>
      <h2>Here's my page</h2>
      <p>World, hello</p>
      <a href="test.html">Link</a>
   </body>
</html>
```

Document Object Model

- The document object itself, as well as all of the objects contained within it, have a number of *properties* and a number of *methods* that can be used to drill down to a very specific piece of your website.
- By resetting those properties or calling certain methods, the contents of our web pages can change without us needing to refresh the page.



Document Object Model

- **DOM Properties**

DOM Property	Description
innerHTML	Holds the HTML inside a set of HTML tags.
nodeName	The name of an HTML element or element's attribute.
id	The "id" attribute of an HTML element.
parentNode	A reference to the node one level up in the DOM.
childNodes	An array of references to the nodes one level down in the DOM.
attributes	An array of attributes of an HTML element.
style	An object encapsulating the CSS/HTML styling of an element.

Document Object Model

- **DOM Methods**

DOM Method	Description
getElementById(id)	Gets the element with a given ID below this point in the DOM.
getElementsByTagName(tag)	Gets all elements with the given tag below this point in the DOM.
appendChild(node)	Add the given node to the DOM below this point.
removeChild(node)	Remove the specified child node from the DOM.

Document Object Model

- If we start from document, we can get to any piece of our web page that we choose, through careful use of DOM properties and methods.

Document Object Model

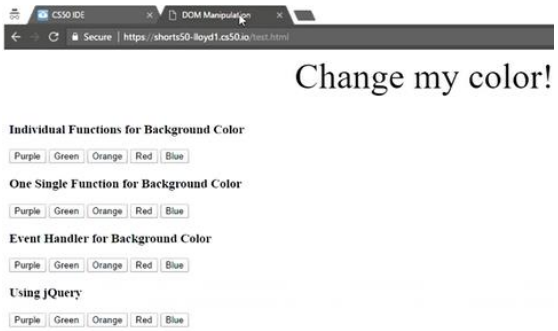
- **jQuery**
- Because DOM manipulation is so common with JavaScript, and because the JavaScript to do so can get quite lengthy, people wanted alternatives.
- jQuery is a popular open-source library, released in 2006, that is designed to simplify client-side scripting (such as DOM manipulations).

Document Object Model

- **jQuery**

~~document.getElementById('colorDiv').style.backgroundColor = 'green'~~

```
$('#colorDiv').css('background-color', 'green');
```




```
<!-- Four different ways of changing colors -->
  <!-- (a) Five separate color-changing functions -->
<script src="js/five.js" type="text/javascript"></script>

  <!-- (b) One general color-changing function -->
<script src="js/general.js" type="text/javascript"></script>

  <!-- (c) An event handler method -->
<script src="js/event.js" type="text/javascript"></script>

  <!-- (d) Using jQuery only -->
<script src="js/jquerycolor.js" type="text/javascript"></script>
```

```
<div>
<h3>Event Handler for Background Color</h3>
    <button onclick="changeColorEvent(event);">Purple</button>
    <button onclick="changeColorEvent(event);">Green</button>
    <button onclick="changeColorEvent(event);">Orange</button>
    <button onclick="changeColorEvent(event);">Red</button>
    <button onclick="changeColorEvent(event);">Blue</button>
</div>
```

```
function turnOrange()
{
    document.getElementById('colorDiv').style.backgroundColor = 'orange'
}
function turnRed()
{
    document.getElementById('colorDiv').style.backgroundColor = 'red';
}
function turnBlue()
{
    document.getElementById('colorDiv').style.backgroundColor = 'blue';
}
```

```
function changeColor(color)
{
    document.getElementById('colorDiv').style.backgroundColor = color;
}
```

```
function changeColorEvent(event)
{
    var triggerObject = event.srcElement;
    document.getElementById('colorDiv').style.backgroundColor = triggerObject.innerHTML.t
}
```

```
triggerObject.innerHTML.toLowerCase()
```

```
<h3>Using jQuery</h3>
    <button class="jQButton">Purple</button>
    <button class="jQButton">Green</button>
    <button class="jQButton">Orange</button>
    <button class="jQButton">Red</button>
    <button class="jQButton">Blue</button>
```

```
$(document).ready(function() {
  $('.jQButton').click(function() {
      $('#colorDiv').css('background-color', this.innerHTML.toLowerCase());
  });
});
```

Flask

- Python is not just used for command-line programming, though that's a major use case.
- Python contains native functionality to support networking and more, enabling site backends to be written in Python.

Flask

- Web frameworks make this process much easier, abstracting away the minutia of Python's syntax and providing helper functions.
- Some of the most popular include: Django, Pyramid, and Flask.
- We use Flask in CS50 because it is lightweight for ease of use in CS50 IDE, while still being feature-rich.

Flask

- We know that we can use HTML to build websites, but websites built using pure HTML suffer from a serious limitation.
- Imagine we want to create a website that displays the current time in Cambridge, MA, displaying it to the latest minute.

Flask

```
<html>
    <head>
        <title>Current Time in Cambridge</title>
    </head>
    <body>
        The current time in Cambridge is 14:08
    </body>
</html>
```

Flask

- Websites that are pure HTML are completely static. The only way we can update the content of our pages is to manually open up our source files, edit and save, and then the next time the user visits or refreshes the page they'll get the content.
- Incorporating Python into our code can make our code quite a bit more flexible and introduce a way for our pages to update or be dynamic without requiring our intervention.

Flask

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return "The current date and time in Cambridge is {}".format(now)
```

Flask

- It's pretty simple to get started using Flask within CS50 IDE.

```
from flask import Flask
```

Flask

- It's pretty simple to get started using Flask within CS50 IDE.
- After importing the Flask module, we need to initiate a Flask application.

```
app = Flask(__name__)
```

Flask

- It's pretty simple to get started using Flask within CS50 IDE.
- After importing the Flask module, we need to initiate a Flask application.
- From there, it's just a matter of writing functions that define the behavior of our application.

Flask

```
def index():
    return "You are at the index page!"


def sample():
    return "You are on the sample page!"
```

Flask

```
@app.route("/")
def index():
    return "You are at the index page!"

@app.route("/sample")
def sample():
    return "You are on the sample page!"
```

Flask

- The lines just added are known as "decorators." They are used, in Flask, to associate a particular function with a particular URL.
- Decorators also have more general use in Python, but that goes beyond the scope of CS50.

Flask

- It's also quite straightforward to run our Flask application within CS50 IDE.

```
export FLASK_APP=application.py
export FLASK_DEBUG=1
flask run
```

Flask

- Data can be passed in via URLs, akin to using HTTP GET.

```
@app.route("/show/<number>")
def show(number):
    return "You passed in {}".format(number)
```

Flask

- Data can be passed in via HTML forms, as with HTTP POST, but we need to indicate that Flask should respond to HTTP POST requests explicitly.

```
@app.route("/login", methods=['GET', 'POST'])
def login():
    if not request.form.get("username")
        return apology("must provide username")
```

Flask

- We could also vary the behavior of our function depending on the type of HTTP request received:

```
@app.route("/login", methods=['GET', 'POST'])
def login():
   if request.method == "POST":
      # do one thing
   else:
      # do a different thing
```

Flask

- Flask has a number of functions within its module that will be useful for application development.

```
url_for()
redirect()
session()
render_template()
```

Flask

- More information available at the Flask quick start guide:

`http://flask.pocoo.org/docs/0.12/quickstart/`

- More information on using Jinja can be found at:

`http://jinja.pocoo.org/`

Ajax

- Up until now, our interaction with JavaScript has been mostly limited to: push a button, something happens.
- We still don't have to entirely reload our page, but there is still some degree of user interaction.
- Ajax (formerly *Asynchronous JavaScript and XML*) allows us to dynamically update a webpage even more dynamically.
  - Though, for now, we won't go too crazy!

Ajax

- Central to our ability to asynchronously update our pages is to make use of a special JavaScript object called an `XMLHttpRequest`.

```
var xhttp = new XMLHttpRequest();
```

Ajax

- Central to our ability to asynchronously update our pages is to make use of a special JavaScript object called an `XMLHttpRequest`.
- After obtaining your new object, you need to define its `onreadystatechange` behavior.
  - This is a function (typically an anonymous function) that will be called when the asynchronous HTTP request has completed, and thus typically defines what is expected to change on your site.

Ajax

- `XMLHttpRequest`s have two additional properties that are used to detect when the page finishes loading.
  - The `readyState` property will change from from 0 (request not yet initialized) to 1, 2, 3, and finally 4 (request finished, response ready).
  - The `status` property will (hopefully!) be 200 (OK).
- Then just make your asynchronous request using the `open()` method to define the request and the `send()` method to actually send it.

Ajax

- `XMLHttpRequest`s have two additional properties that are used to detect when the page finishes loading.
  - The `readyState` property will change from from 0 (request not yet initialized) to 1, 2, 3, and finally 4 (request finished, response ready).
  - The `status` property will (hopefully!) be 200 (OK).
- Then just make your asynchronous request using the `open()` method to define the request and the `send()` method to actually send it.
  - There is a slightly different way to do this syntactically with jQuery!

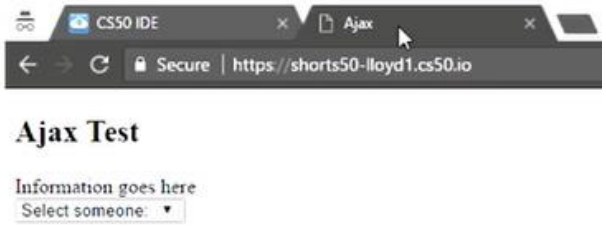
Ajax

```
function ajax_request(argument)
{
    var aj = new XMLHttpRequest();
    aj.onreadystatechange = function() {
        if (aj.readyState == 4 && aj.status == 200)
            // do something to the page
    };

    aj.open("GET", /* url */, true);
    aj.send();
}
```

Ajax

- More commonly, you'll see Ajax requests written using jQuery instead of "raw" JavaScript.

http://api.jquery.com/jquery.ajax/



#1 JavaScript tutorial for beginners
#2 variables
#3 arithmetic expressions
#4 user input
#5 type conversion
#6 const
#7 Math
#8 hypotenuse calc practice program
#9 counter program
#10 random number generator
#11 useful string methods
#12 string slicing
#13 method chaining
#14 if statements
#15 checked property
#16 switches
#17 AND OR logical operators && ||
#18 NOT logical operator
#19 while loops
#20 do while loops
#21 for loops
#22 break and continue statements
#23 nested loops
#24 functions
#25 return statement
#26 ternary operator
#27 var vs let
#28 template literals
#29 format currency
#30 number guessing game

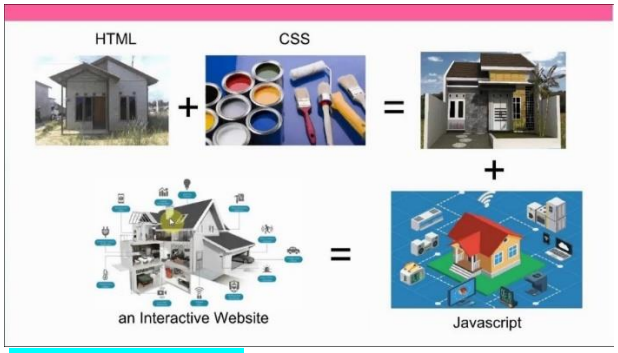
#31 temperature conversion program
#32 arrays
#33 loop through an array
#34 sort an array of strings
#35 2D arrays
#36 spread operator
#37 rest parameters
#38 callbacks
#39 array.forEach()
#40 array.map()
#41 array.filter()
#42 array.reduce()
#43 sort an array of numbers
#44 function expressions
#45 arrow function expressions
#46 shuffle an array
#47 nested functions
#48 maps
#49 objects
#50 this keyword
#51 classes
#52 constructors
#53 static keyword
#54 inheritance
#55 super keyword
#56 getters & setters
#57 objects as arguments
#58 array of objects
#59 anonymous objects
#60 error handling

#61 setTimeout()
#62 setInterval()
#63 Date objects
#64 clock program
#65 asynchronous
#66 console.time()
#67 promises
#68 async
#69 await
#70 ES6 Modules
#71 DOM intro
#72 element selectors
#73 DOM traversal
#74 add/change HTML elements
#75 add/change CSS properties
#76 events
#77 addEventListener()
#78 show/hide HTML elements
#79 detect key presses
#80 animations
#81 canvas API
#82 window
#83 cookies
#84 stopwatch program
#85 rock paper scissors game
#86 tictactoe game
#87 snake game
#88 pong game

89. strict equality ===
90. hoisting
91. concat()
92. .join()
93. Array/Objects Methods
94. Add a Read More Hidden-Text Drop-Down in a simple JavaScript, HTML and CSS
95. anonymous functions
96. Closure
97. Closure examples

1. JavaScript tutorial for beginners

****




```
adding border to DIV= border:
div{
1px solid red !important;
}
div:after{
content:attr(class);
}

//alert("AI');
console.log("Advanced AI");

/*
multi line comment
*/

//Text Editors: vs code, sublime text, atom, notepad
//download node.js
//node.js = runtime environment that runs on chrome V8 engine and executes JavaScript code outside a web browser
//you can use node.js to install third-party libraries
//download from nodejs.org
//you put the script tags at the end because you want the HTML and CSS load first then JavaScript also in case something happens to JS then you will have a blank page
//so the script tags should be at the end of the body of your document
//at the beginning of the script tag you add the attributes src for your source and in the quotes, we will put the name of the JavaScript file that we're going to link//
```

//

2. Variables

****

```
//variables = unique identifier for some value
//variables can be used to represent Boolean value, words, number, objects

//creating variables is done in 2 steps:
1.declaration
2.assignment

//we create variable for ex to hold a number

//example
//we declare var by typing var
//we have unique identifier which is age
//we are going to store a number in var age
//JavaScript has a feature named automatic semicolon insertion which
can insert semicolons behind scenes.

var age=21;
var whatever;

age=age+1;
```

```
console.log("your age is:), age);

//camelCase = capitalize all letters of var after the first one
//string = one or more characters

var firstName="Bro";
var lastName="Sergiulica";
var myCoolVariable

console.log("your first Name is"+firstName);

//var Boolean is like a light switch, either on or off

whatever = "code";

var online=true;
console.log("Are you online?:", online);
console.log(whatever);

var fullName=firstName+" "+lastName;
console.log(whatever);

var fullName=firstName+" "+lastName;
console.log("Hello, fullName);

//there are 2 additional options to declare a variable
//one of them is let
//with let you assign temporary variable a limited scope

//let mySweetVariable

//2nd method is to use const which stands for constant
//so Variables are a container for some sort of value
//and they behave as the value that they contain
//you can store more than just numbers, strings. Booleans
```

//

3. Arithmetic expression

****

```
/*arithmetic expression:
is some combination of operands(values,variables, etc.)
operators (+ - * /)
that can be evaluate to a value
*/
//you can raise a variable or other value to a power by**
//modulus % = remainder of any division, you need a divisor and dividend

//var friends=10;
//friends=friends +1;
//friends=friends -1;
//friends=friends *2;
//friends=friends /2;
//friends=friends ** 2;
//var remainingFriends=Friends %3;

//friends +=1;
//friends -=1;
//friends *=2;
//friends /=2;
//friends **=2;
//friends %=3;

//console.log(remainingFriends);
//console.log(friends);
```

```
/*
operator precedence <<-multi arithmetic expression< so you have to go in sequence.
1.parenthesis()
2.exponents
3.multiply + & divide /
4.addition + & subtraction -
acronym: pemdas= please excuse my dear aunt sally
you can force the operator precedence by surrounding part of the equation with parentheses.
*/

friends=(10 + 2) / 2;
console.log(friends);
```

//

4. HTML user input

****

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title Document</title>
</head>
<body>
<label>Enter your name:</label><br>
<input type="text" id="mytext"<br>
<button type="button" id="myButton">submit</button>

<script src="user input.js">
</script>

</body>
</html>
```

```
############
JS user input
############

//The easy way//
//we will ask user to type the name
//var myName=window.prompt("enter your name?");
//console.log("hello", myName);

//The hard way//
//() = it is a method
//button will have the propriety on click so it will do something
//by typing document, you create reference of the html document
//function=perform a task for us when it is called
//value=we add value because we need to access to this value
//after a method, an action will come

document.getElementId("myButton").onclick=function(){
var myName=document.getElementbyId("myText").value;
console.log("hello",myName);}
```

```
// import readline module
const readline = require("readline");

// create interface for input and output
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

// create empty user input
let userInput = "";
```

```
// question user to enter name
rl.question("What is your name\n", function (string) {
  userInput = string;

  console.log("Your name is " + userInput);

  // close input stream
  rl.close();
});
```

```
   const readline = require('readline').createInterface({
   input: process.stdin,
   output: process.stdout
   });

   readline.question('Who are you?', name => {
   console.log(`Hey there ${name}!`);
   readline.close();
   });
```

```
const readline = require('readline')

const inquirer = readline.createInterface({
   input: process.stdin,
   output: process.stdout
});

inquirer.question("What is your name ?", name => {
   inquirer.question("How old are you?", age => {
       console.log(`${name} is ${age} years old`);
       inquirer.close();
   });
});

inquirer.on("close", function() {
   console.log("Good bye!");
   process.exit(0);
});
```

//

5. Type conversation

****

```
//Type conversion of numbers, strings, Booleans
//(explicit vs implicit)

//change the datatype of a value to another

//var age=window.prompt("Enter your age");
//console.log("Happy Birthday!");
//age=Number(age);
//age +=1;
//console.log(typeof age);
//console.log(age);

myVar=Boolean("pizza");

console.log(typeof myVar);
console.log(myVar);
```

//

7. Math

*****

```
//Math = an object that provides basic mathematics functionality and constants

var myNum;

//myNum=Math.round(3.5);
//myNum=Math.floor(3.9);
//myNum=Math.ceil(3.1);
//myNum=Math.pow(3.2);//ridicare la putere
//myNum=Math.sqrt(64);
//myNum=Math.abs(-1);
//myNum=Math.min(1.5.2.4.3);
//myNum=Math.max(1,5,2,4,3);
//myNum=Math.PI;
//myNum=Math.E;
//console.log(myNum);

//solve for hypotenuse of a right triangle
//c = radical din a la 2a + b la 2a

var a = window.prompt("enter side A");
var b = window.prompt("enter side B");
var c;

c=Math.sqrt(Math.pow(a,2)+Math.pow(b,2));
console.log("Side c=",c);

//Math.random()
const Tetrominoes=['Ltetromino','ZTetromino','iTetromino']

//console.log(Math.random())//random 0 < 1
//console.log(Math.random()*20)//random 0 <20
//console.log(Math.random()*Tetrominoes.length)

const randomNumber=Math.random()*Tetrominoes.length

//console.log(randomNumber)

//Math.floor(rounds number to nearest full integer)
//Math.ceil() - rounds up
//Math.round-round to nearest integer

const randomInteger=Math.floor(randomNumber)
//console.log(randomInteger)
console.log(Tetrominoes[randomInteger])]

###################
 ex. choices of drinks:
###################

const drinksMenu=['vodka Tonic','white wine','beer','mocktail']

const randomDrink=Math.random()*drinksMenu.lenth

const friendsChoice=(Math.floor(randomDrink))

console.log(drinksMenu[friendsChoice])

//or

console.log(drinksMenu[Math.floor(Math.random()*drinksMenu.length)])
```

//

10. Random numbers

****

```
//Math.random()=generates a pseudo-random between 0 and 1 and NOT true numbers
//don't use this for security purposes but for games it's ok

var randNum;
//randNum=Math.random();//0-1
//randNum=Math.random() * 6;//random number between 0-5
//randNum=Math.floor(Math.random() * 6);//random number between 0-5 round up
//randNum=Math.floor(Math.random() * 6+1;//random number between 0-6 round up

//You can also make a function to generate a random number for us and call it whatever we want
function getRandomNum(min,max){//2 parameters for our range
return Math.floor(Math.random()*(max-min+1)+min)
}

//we call it 3 times for 3 times dice
randNum=getRandomNum(1,6);
console.log(randNum);

randNum=getRandomNum(1,6);
console.log(randNum);

randNum=getRandomNum(1,6);
console.log(randNum);
```

//

11. String methods

****

```
//useful string properties & methods

var myName="Bro Code";
var myStreet="123 Fake st.";
var myCity="Nowhere XY 123456";
var myPhone="123-456-78890";

console.log(myName.length);
console.log(myName.charAt(2));
console.log(myName.indexOf(" "));
console.log(myName.lastIndexOf("o"));
console.log(myName.trim());
console.log(myName.toUpperCase());
console.log(myName.toLowerCase());

var firstName=myName.slice(0,3);
var firstName=myName.slice(0,myName.indexOf(" "));
console.log(firstName);

var lastName=myName.slice(myName.lastNameIndexOf(" ")+1);
console.log(lastName);

var myAddress=myStreet.concat("",myCity);
console.log(myAddress);

myPhone=myPhone.replaceAll(-","");
console.log(myPhone);
```

//

13. Method chaining

****

```
//method chaining = calling one method after another, in one continuous line of code
//example: capitalization

//user.trim().charAt(0).toUpperCase().whatever()

//example we will capitalize a username with and without method chain
```

```
let user="bro";
//let firstLetter=user.charAt(0);
//firstLetter=firstLetter.toUpperCase(0);
//userr=firstLetter+user.slice(1);

user=user.charAt(0).toUpperCase()+user.slie(1);

console.log(user);
```

//

14. If statements

****

```
//if statement= a basic form of decision making if a condition is true, then do something if not, then don't do it!
//you can add more than 1 if by else if and last one is else
//it is important also the order of the statements or it will not work

var age=5;

if(age>=65){
console.log("you are a senior!");
}
else if(age>=18){
console.log("you are an adult");
}
else if(age < 0){
console.log("you are not born yet);
}
else {
console.log("you are a child");
}

//example of somebody is online

var online = true;
if(online==true){//or shortcut if(online){//this works only for boolean
console.log("Bro is online");
}
else{
console.log("Bro is offline");
}
```

//

16. Switches

****

```
//switch = statement that evaluates a value/expression
//matches the value to many case clauses
//more efficient that many "else if" statement

//var grade ="A";

//if(grade=="A"){
//console.log("You did great!");
//}
//else if (grade=="B"){
//console.log("you did okay!");
//}
//else if (grade=="D"){
//console.log("You..did not do that well");
//}
//else if (grade=="F"){
//console.log("You have failed");
//}
//else{
//console.log(grade,"is not a letter grade");
```

```
//}

//Same thing can be written as:
//keep var as it is

var grade="Pizza";

switch(grade){
case "A":
console.log("You did great!");
break;
case "B":
console.log("You did okay !");
break;
case "D":
console.log("You..did not do that wel");
break;
case "F":
console.log("You have failed");
break;
default:
console.log(grade,"is not a letter grade");
}
```

//

17. Logical operators

****

```
//logical operators = used to write more complex conditions

// && and (checks if both conditions are true)
// || or (checks if at least 1 condition is true)
//! not (reverses boolean value)

//&&
//ex we will write program to check temperature of the room
//display also a custom message based on temperature
//we will have a var temperature and we will set this to 15 degrees

//var temperature = -10;//Celsius
//if)temperature > 30){
//console.log("It is Hot outside!");
//}
//else if(temperature > 0 && temperature <=30){
//console.log("It is warm outside!");
//}
//else{
//console.log("It is cold outside");
//}

//||
//ex we will write a program to check if it's light outside or dark based on the hour

//var hour=8;//military time
//if (hour < 6 || hour >=20){
//console.log("it's dark outside!");
//}
//else{
//console.log("It's light outside!");
//}

//!
//ex accept some user input and then ask user to type in the name
//if somebody will click ok it will sign an empty string to my var

var myName=window.prompt("Enter your name");
if (!(myName=="")){
```

```
console.log("Hello", myName);
}
else{
console.log("you did not enter your name!");
}

//the not comparison vs logical operator as above
//if(myName !=""){
//console.log("Hello",myName);
//}
//else{
//console.log("You did not enter your name");
//}
```

//

19. While loop

****

```
//while loop = repeats same code, while the loop's condition remains true
//used for tech support scams for pop up with message that computer has a virus and enter credit card info
//creditNum for credit card number
//variation of while loop: do loop //do while loop
//with do while we do not check the condition until the end of the while loop, so it will check the block once then our condition to see if it's true

//var credit Num=window.prompt("Your computer has a virus!Enter your credit card info...or else!");

//while(creditNum == ""){
//creditNumer=window.prompt("your computer has a virus!!Enter your credit card info..or Else");
//}

do{
var creditNum=window.prompt("your computer has a virus!!Enter your credit card info..or Else");
}while(creditNum=="")

console.log("Thank you, your computer no longer has virus");
```

//

21. For loops

****

```
//for loop = repeat a loop a limited number of times for (declare index; condition; step)
//it is like a while loop but it does not repeat infinitely
//ex we want to create a program to count till 10
//so you have for(){} and in the parenthesis  you will have 3 optional statements
//first one will be declaring the index as how many times we want to repeat for loop
//second is the condition
//last one is how much we want to increment our index after each iteration
//so next loop will be executed 10 times
//and will stop when the condition will not be true

/*
for(let i=1; i<=10;i++){
console.log(i);
}
*/

//counting by two use i+=2

//ex of counting backwards for new year and displaying a message

/*
for(let i= ; i>0;i--){
console.log(i);
}
*/
```

```
//or you can write it like this but do not forget about ; or it will not work

let i=;

for(;i>0;){
console.log(i);
i--;
}
console.log("Happy New Year");
```

//

22. Break & continue

****

```
//break = used to exit out of a loop
//continue = skips over an iteration of a loop

/*
for(let i=1;i<20;i++){
if(i==13){
break;
}
console.log(i);
}
*/

for(let i=1; i<=20;i++){
if(i==13){
continue;
}
console.log(i);
}
```

//

23. Nested loops

****

```
//nested loop=a loop inside of another loop
//ex printing 2d matrix of symbols to a webpage
//we will need a user input
//outer loop will be in charge of the rows
//usually with nested loops after i you use j

var symbol=window.prompt("Enter a symbol to use");
var rows=window.propmt("Enter # of rows");
var columns=window.prompt("Enter # of column");

for(let i=0; i<rows;i++){
for(let j=0; j<columns;j++){
//console.log(symbol);
document.getElementById("myRectangle").innerHTML+=symbol;
}
document.getElementbyId("myRectangle").innerHTML+="<br>";
}

Obs.In HTML add <h1 id="myRectangle"></h1>
```

//

24. Functions

****

```
/*
A function is a block of code designed to perform a task/procedure.(subroutine).

We invoke "call" whenever we want it to do something.
We need to define what we want to ask to do but first we need to define what our function has to do.
*/
```

```
//first we type function then the name of the function
//after ass parenthesis, semicolon and ; = () {} ;
//{} is the body of our function so when we call the function it will execute whenever is inside{}
//to call this function we type the name and ()
//you can write a whole line of code needed inside {}
//functions have access to any variables outside the function
//functions have access to var as it is known global scope
//anything declared inside of a function is considered to have a local scope
//you have global and local scope
//so functions have access to anything that is global
//you can always pass values or variables to a function
//so when you call a function you just list a value or a variable here
//when you send values to a function these are known as arguments but do not confuse them with parameters
//you will need a matching parameters
//so you either can either use variables or parameters = myName in function has to match what is on the console
//with function you can return also a value -- return --

//function sayHello(){
//console.log("Hello");
//console.log("Have a nice day!");
//console.log("Goodbye!");
//};

//sayHello();

//function sayHello(myName,myAge){//parameters
//console.log("Hello", myName);//local scope
//console.log("You are", myAge,"years old")
//};
//var myName="Bro";//global scope

//sayHello("Bro",21);

//using return we will make a program to convert fahrenheit to celsius//
//so we will create a function that will create that for us//
//formula to convert is (32 degreesF-32)x5/9 so (f-32)x5/9

functiontoCelsius(f){
//var result=(f-32)*(5/9);
//return result;

return (f-32)*(5/9);//shortcut without storing into a variable like result
};
//formula for farenheit into C is : (0 degrees C x9/5)+32
function to Fahrenheit(c){//the parameter is c
return (c*9/5)+32;
};
toCelsius(100);//this won't do anything as the value is returned up
//so we use a var and store it there

//var myTemp=toCelsius(100);
//console.log("My Temp in C is: ",myTemp,"degrees");//display

var myTemp=toFarenheit(37.7);
console.log("My temp in F is:,myTemp,"degrees");//display

const bob=document.querySelector('.class')
let count=0
function moveBob(){
count+=50
console.log(count)
bob.style.left=count+'px'
}
bob.addEventListener('click', moveBob)

const balls=document.querySelectorAll('.ball')
console.log(balls.length)
```

```
//CZ: a function is a block of code that is designed to perform a particular procedure or task whenever it is called that function
//you can pass a var or value o a function and that is called arguments but with matching parameters
```

//

26. Ternary operator ?

****

```
//ternary operator = shortcut for if statements
//takes 3 operands
//1.a condition with ?
//2.expression if True :
//3.expression if False

//condition? exprIfTrue : exprIfFalse

//example 1 to check the winner:
//function checkWinner(win){//win is the parameter that will store boolean of true or false
//win ? console.log("you win!") : console.log("you loose"!);
//}

//checkwinner(true);

//example 2 to return a value, check age at the bar

function checkAge(age){
return age >=21 ? "you get beer!" : " you got apple juice"
}

console.log(checkAge(21));
```

//

27. var vs let

****

```
//ECMAScript (ES6)
//"JavaScript standard meant to ensure the interoperability of web pages across different browsers"
//scope = where a variable is accessible

//let in block & var in function
//let=declare variables with a block scoped{}
//var=declare variables with a function scoped()

//function doSomething(){
//for (var i=1;i<=3;i++){
//console.log(i);
//}
//}
//doSomething();
//console.log(i);//this will give error//if you change var then no error

//var name="";
let name="Bro Code";
```

//

32. Arrays

******

```
//array = special variable that can hold more than one value

//Each"space" is known as an element
//you access elements in an array by referring to the index number ex.[0]
//for example you can create array for garage or parking lot
//var then name of array that will hold the names of the car
//all will be stored within []
```

```
//separate each array with ,
//if you need to access a certain value or element of the array you need to list the index number as well
//each spot or space is called element (like parking spot and you need to list the parking number)
//first element will be 0
//use push() to add element to the array
//use pop() to remove last element
//if you need to see the length, how many values are stored within an array you can use the length property of arrays so you type the name of the array.length
//assign that length to a new variable

var cars=["Mustang","Audi","Jaguar"];

//console.log(car[0]);
//console.log(car[1]);
//console.log(car[2]);

//console.log(car[3]);

//cars.push("Tesla");//add an element
//cars.pop();//removes last element

//cars=cars.sort();//sort out array alphabetically
//console.log(cars);
//cars=cars.reverse();//reverse sort an array
//console.log(cars);

var numberOfCars=cars.length;
console.log(numberOfCars);

//var lastCar=cars[cars.length-1];

//console.log(lastCar);
```

//

33. For in loop

****

```
//for in loop=loop through the elements in an array or the properties of an object
//less syntax to write, but less flexible

let letters=["h","e","l","p"];

/*
for(let i=0; i<letters.length;i++){
console.log(letters[i]);
}
*/

//for(let i in letters){
//console.log(letters[i]);
//}
//so for in loop is less flexible than the traditional loop
//you cannot decrement or select different increments
//the for in loop we can loop through the properties of an object

let car={
make:"Chevy",
model:"Corvette",
year:2022,
color:"blue"
}

for(let property in car){
console.log(car[property]);
}
```

//

38. Callback

****

```
//callbacks=function passed as an argument to another function
//allowed a function to invoke another function

//example I want to pass a message either to console or HTML window

let message;

function displayConsole(output){
console.log(output);
}

function displayDOM(output){
document.getElementById("myH1").innerHTML=output;
}

function setMessager(text,callBack){
message=text;
callBack(message)
}

setMessage("Hello", displayDOM)
```

//

39. array.forEach()

****

```
//forEach()=performs a function for each element in an array
//Let's say we have a store and we need to calculate the total after somebody adds a bunch of items in their shopping cart

let total = 0;
let cart=[5,6,7,8,9];

function checkOut(element, index, array){
total+=element;
}

cart.forEach(checkOut);
console.log("Your Total is:$"+total);
```

//

40. array.map()

****

```
map() = performs a function for each element in an array, then stores the returned values in a new array
//imagine that you have an online store that have below different prices
//we want to convert all those prices in euros

let storeUSD=[5,6,7,8,9];

//we need to define a function that changes USD to Euro with at least one parameter as value

function to EUR(value){
value *=0.85;
return value;
}

let storeEUR=storeUSD.map(toEUR);
console.log(storeUSD);
console.log(storeEUR);
//so it will use function for each array and return in a new array
```

//

41. array.filter()

****
//filter()=returns the element of an array that meets a condition specified in a function

```
let sudents=[16,17,18,19,20];

function checkAge(age,index,array){
if(age>=18){
return age;
}
}

let adultStudents=students.filter(checkAge);

console.log(adultStudents);
```

//

42. reduce()

****
//reduce()=reduces an array to a single value
//the return value of the callback function is the accumulated result, and is provided as an argument in the next call to callback function.

```
//pretend you are playing a game of scrabble

let letters=["H","E","L","P"];

//we will need a function to combine those letters together

function combineLetters(total, nextLetter,index,array){
return total+nextLetter;
}

let word=letters.reduce(combineLetters);

console.log(word);
```

//

44. Function expressions

****
//function expression=A function can also be defined using an expression
//Function expression can be stored in a variable.
//The variable can be used as a function.
//They can be invoked by adding ()

//1.Useful as closures
//2.Used in IIFE(IIFE=immediately invoked functions=funcs that runs as soon as it's defined)
//3.Can be passed as arguments to other functions

```
####################
 function declaration
####################

//sayHello();
//function sayHello(){
//console.log("Hello")
//}

###################
 function expression
###################

//let greeting=function(){console.log("Hello")}
let greeting=function(){document.getElementById("myH1").innerHTML=("Hello")}
output(greeting);
function output(anyName){
anyName();
```

```
}
```

//

47. Nested functions

****

```
//outer functions have access to inner functions.
//Inner functions are "hidden" from outside the scope.
//used in closures

function login(){
let userName="Bro";
let userInbox=0;

function displayUserName(){
console.log("Hello", userName);
}
function displayUserInbox(){
console.log("You have",userInbox,"new message");
}
displayUserName();
displayUserInbox();
}
login();
```

//

49. Objects

****

```
//objects = a container for properties, methods

//property = values/variables that describe what an object has
//methods = functions that describe what an object can do

//through properties and methods, we can create objects that mimic things in real world programming

//ex we are going to create human object
//human has properties and methods
//properties are the values variables that describes what an object has like name, age
methods are functions that belong to objects=describe what an object can do like humans they eat, it's kind of like a verb so what actions the humans can perform and the objects can perform it whenever we call it
//creating objects, it is like creating a variable
//ex var human then {} then ;
//what is on the {} is owned by our human object
//add , not ; if you want to add more functions to the object
//we create an eat,drink,sleep functions
//last } you do not need ,
//to access the object you use console.log(human); or human.
//you need to access specific properties of the object so you can use dot or bracket notation

var human1={
name:"Rick",//property
age:65,//property
eat: function(){//function
console.log("Rick is eating food")
},

drink: function(){
console.log("Rick is drinking alcohol *burp*")
},

sleep:function(){
console.log("Rick has passed out")
}
};

//human1.name//dot notation
```

```
//human1['name']//bracket notation

console.log(human1.name);
console.log(human1.age);
human1.eat();
human1.drink();
human1.sleep();

var human2={
name:"Morty",//property
age:16,//property
eat: function(){//function
console.log("Morty is eating food")
},

drink: function(){
console.log("Morty is drinking water")
},

sleep:function(){
console.log("Marty is asleep")
}
};

//human2.name//dot notation
//human2['name']//bracket notation

console.log(human2.name);
console.log(human2.age);
human2.eat();
human2.drink();
human2.sleep();
```

//

63. Date objects

****

```
//date object = represents a moment in time epoch(reference point)

//this is to explain date objects
//below will process the date when the computer thinks the internet was created
date = new Date(0);//if you change 0 it will give you the date of today
date = new Date();
//to get current date then pass no argument

//construct a unique object
//pass in 7 arguments=year,month,day,hour,pm,minutes,2 seconds,3 milliseconds
//date=new Date(2022,3,4,18,1,2,3);

//or

//string representation of date and time
date=new Date("April 20,2022,16:20:01:02");

/*
//console.log(date);

//use get methods
let year = date.getFullYear();//get full year method
let month=date.getMonth();
let dayOfWeek=date.getDay();
let dayOfMonth=date.getDAte();
let hours = date.getHours();
let seconds=date.getSeconds();
let miliSeconds=date.getMilliseconds();
//let month=
//let dayOfWeek=
```

```
//let hours
//let minutes=
//let seconds
//let milliseconds

console.log(year);//so my variable year has a value of 2022
console.log(month);//January 0
console.log(dayOfWeek);//Sunday is 0
console.log(hours);
console.log(minutes);
console.log(seconds);
console.log(milliSeconds);
*/

date.setFullYear(2033);
date.setMonth(11);
date.setDate(25);
date.setHours(0);
date.setMinutes(0);
date.setSeconds(0);

console.log(date);

//Cz:a date is a representation of a moment in time
//epic is our reference point

    //Date Objects
    const today=new Date();
    const Display=today.getDate()//the day

    const day=today.getDay()//the day of the week

    console.log(day)
    const dateDisplay=document.querySelector('.red-circle')

    const fullYear=today.getFullYear()
    console.log(fullYear)

    const hours=today.getHours()

    console.log(hours)

    const isoString=today.toISOString()//used by programers
    console.log(isoString)

    //ex.count down till Christmas

    const Christmas=new Date('2021-12-24T13:54:14.172Z')
    console.log(Christmas-today)//will give in milisecounds

    const millescoundsToXmas=Christmas-today
    const minutes=Math.round(millsecoundsToXmas / 60000)
    dateDisplay.textContent=minutes
    console.log(minutes)
```

//

72. 3 ways of picking up elements from HTML

```
**********
1).querySelector()-pick up elements from HTML so pick specific like class
const circle=document.querySelector('circle')
.querySelectorAll-pick up more than 1 element

2)grabing querySelector with id
const circle=document.querySelector('#main')

3)you also use getElementById
```

```
const circle=document.getElementById('main')

console.log(circle)
```

//

77. .addEventListener()
****

```
//target.addEventListener(event, function)

const test=document.querySelector('.circle')
function alert(){
console.log("clicked")
}
test.addEventListener('mouseover',alert)

obs.if you put script in html in the head at the begining then you have to write this in JS:
document.addEventListener('DOMContentLoaded', ()=>{
//write here the code
})

const body=document.querySelector('body')

body.addEventListener('click', function(){console.log('clicked')})

##########################################
Making ball changing color with addEventListener
##########################################

###
css
###
body{
font-family:sans-serif;
background-color:orange;
display:flex;
justify-content:center;
margin-top:80px;
}

.circle{
width:200px;
height:200px;
border-radius:100px;
background:blue;
}

.red-circle{
width:200px;
height:200px;
border-radius:150px;
background:red;
}

##
JS
##
const circle = document.querySelector('.circle')
function toggleColor(){
circle.classList.toggle('red-circle')
}
circle.addEventListener('click',toogleColor)
//So by click it will toggle between the div and the css re-circle (adding-subtracting)

####################
Remove EventListener
####################
```

```
const circle = document.querySelector('.circle')
let count=0

function toggleColor(){
circle.classList.toggle('red-circle')
count++
console.log(count)
if(count>10){
circle.removeEventListener('click',toggleColor)
}
}
circle.addEventListener('click',toogleColor)

##################################
create moving ball using EventListener
##################################

const circle=document.querySelector('.scrcle')
let height=0
function moveCircle(){
height +=100
circle.style.top=height+'px'
if(height>300){
circle.removeEventListener('click',moveCircle)
}
}
circle.addEventListener('click',moveCircle)
```

//

89. strict equality ===

****

```
//loose equality operator ==
//strict equality operator ===

// check for equal value AND type
// compares if value is equal &
// if the data type is equal

//var pi=3.14;//number
//var pi="3.14";//string
//even if they have the same value they are from different data type

if(pi===3.14){
console.log("yes, that is pi");
}
else{
console.log("no, that is not pi");
}
//so loose equality disregards the data type but the strict equality will not disregard it
```

//

90. hoisting

****

```
//hoisting = JS's default behavior of moving declarations, to the top of the current scope.

//A variable can be used before it has been declared.
//Beneficia to understand to help avoid bugs.

//Good practice to declare all variables
//at the top of every scope.

 ##############
   Declaration
 ##############
```

```
myName="Bro";
console.log(myName);

//var myName;//js will put var at the beginning as if you would have declared it from the start
//let myName;//with let and const you will have error

############
  Initialize
############

var firstName="Bro";
console.log(firstName,lastName);
var lastName="Code";
```

//

91. concat()

****

```
const string='say cheese, '
const string2='AI'

cnsole.log(string.concat(string2))

const array1=[1,2,3]
const array2=[4,5,6]
const array3=[7,8,9]

const newArray=console.log(array1.concat(array2,array3))

consople.log(newArray)
```

//

92. join()

*****

```
//returns a new string

const textDisplay=document.querySelector('.test')
const emotions=['happy','sad','confident']
//textDisplay.innerHTML='test'

const newWord=emotions.join(-)
textDisplay.innerHTML=newWord
```

//

93. Array/Objects Methods

****

```
.pop() -erase last
.shift()-erase first
.unshift() -add beginning
.push() -add at end
.slice() - does not change it produce new ones
.splice()
.forEach()-change in the same time for all elements
X.some() -verify array for any element and return true/false
X.map() - creates new array
X.filter() -creates a new array if rules pass the code(filtering)
X.reduce()-add up what is in the array
X.every()-if every array element matches then return true
X.split()-takes text and splits words as array

//changes the length of the array

const display=document.querySelector('display')
```

```
const movies=['avatar','aliens','fear']

movies.pop()//removes last item from an array

display.innerHTML=movies

display.forEach(anyName.style.backgroundColor='red')

names.forEach(name=>console.log(name))

scores.forEach(score=>console.log(score+2))

console.log(scores.some(score=>score>50))

const ages2018=[21,45,56,45,2]
const ages2021=ages2018.map(age=>age+3)_
console.log(ages2021)

const scores=[3,5,6]
//scores.reduce((accumulator,currentValue)=>accumulator+currentValue)
const total=scores.reduce((accumulator,currentValue)=>accumulator+currentValue)
console.log(total)

const testResults=[45,34,76,89,45,65,67]
console.log(testResults.every(result=>result>30))

const sentence='I will have what she is having'
const words=sentence.split(' ')
console.log(words[2])

############
Array Work!
############

//Using 'map' write a function that converts fahrenheit to celsius.
//eg. getCelsius([23,140,212,41])=>[-5,60,100,5]
//(num -32)x5/9=0C
//const f=[23,140,212,41]
//function getCelcius(){
//return f.map(num=>(num-32)*5/9)
//}
//console.log(getCelcius())
//so returns the value of num

//Using 'some' write a function that checks an array for a 'falsely'value
//eg.checkForFalsely([11,NaN,[],'Angels']
//!=falsely
//const array=[11,NaN,[],'Angels']
//function checkForFalsely(){
//return array.some(verify=>!verify)
//}
//console.log(checkForFalsey())
//so some checks if some of the items in the array match certain rule
//then we will have a boolean response

//Using 'reduce' write a function that will return the total number of characters in an array of strings.
//eg. getTotal([Rabbit','Football','Cracking'])=>22
//const words=['Rabbit','Football','Cracking']
//function getTotal(){
//return words.reduce((total,word)=>total+word.length,0)
//}
//console.log(getTotal())
//so reduce counts all array number or letters

//Using 'every' write a function that checks whether every number in an array is a square number 9 so 3*3)
//e.g checkSquares([9,16,81])=>true
```

```
//const numbers=[9,16,81]
//function checkSquares(){
//return numbers.every(number=>Math.sqrt(number)%1===0)
//}
//console.log(checkSquares())
//so every checks if it is true or false every element

//Using 'filter' as an array method, write a function that returns the elements of an array that have a given length or larger
//e.g getWords(['Florida','dog','phone'],5=>['Florida','phone']
//const wordsArray=['Florida','dog','phone']
//const number=5
//function getWords(){
//return wordsArray.filter(word=>word.length>=number)
//}
//console.log(getWords())
//so filter the array based on the rule inside

//Using 'map' as an array method, write a function that converts an array of cm values as strings, into an array of numbers
//e.g getValues(['23cm',5.6cm','1000cm'])=>[23,2.6,1000]
//const string=['23cm',5.6cm','1000cm']
//function getValues(){
//return string.map(word=>parseFlaot(word))
//}
//console.log(getValues())
//can be written like this:
//function getValues(array){
//return array.map(value=>parseFloat(value))
//}
//console.log(getValues([34cm',45cm','1.2cm'])
//console.log(getValues([3cm',4cm','7cm'])
//so map was used to create a new array based on an instruction

//Using 'split' and 'filter' write a function that counts the number
//of the vowels in a sentence.
//eg. getVoelCount('In West Philadelphia, born and raised')=>12
//const text='In West Philadelphia, born and raised'
//const vowels=['a','e','o','u','i','A','E','O','U','I']
//function getVowelCount(sentence){
//return sentence.split('').filter(letter=>vowels.includes(letter)).length
//}
//console.log(getVowelCount("I m a dog"))

//Using 'split' , 'map' and 'join' write a function that capitalizes the first letter of each word in a sentence
//e.g capitalise()=> 'The Queens Gambit'
//const text='the queens gambit'
//function capitalise(title){
//return title.split(' ').map(letter.charAT(0).toUpperCase()+letter.substr(1)).join(' ')
//}
//console.log(capitalise('life of pie'))
```

//

94. Add a Read More Hidden-Text Drop-Down in a simple JavaScript,HTML and CSS

****

```
//inbuilt JavaScript functions used
-addEventListener()
-querySelectorAll()
-getElementById()
-dataset.target
-innerHTML
-classList.toggle
-forEach()
-Arrow Functions

##########
HTML used
#########
```

 generic inline container for phrasing content, only includes global attributes
Some Text
ex. See my portofolio

#######
CSS used
#######

overflow: specify whether to clip content or to add a scrollbars when content is too big to fit in the specified area, as values(hidden,scroll,auto - same as scroll but it adds scrollbars when needed)
Transition: allow to change property values smoothly, over a given duration, you need to specify 2 things, the CSS property and duration
display: property specifies the display behavior of an element
inline: display inline like
block: display element of a block like

############
HTML Coding
############

```
<!DOCTYPE html>
<html>
<title>Show More</title>
<Link rel="stylesheet" href="project.css"></link>

</head>

<body>

<div class="card">
<div class="placeholder-for-image">
</div>
<div class="info">
<h2>AI</h2>
</div>
<div class"description">
<div class="expandMoreContent" id="showMoreContent1">
<p>
kffkjdj kdjfdjfkjd fs\djf dsfjjfkdsj djfkjsdkj
kfjkdjfkdsjfkjdkf kdj djkdjkfjkdjf dikldfj df
dj fkdjfkjd kjfjdkfjkjfjfkjdfjkdj fkjd f dj fkjd
</p>
</div>
<div class="expandMoreHolder".>
<span expand-more data-hidetext="Show Less..." data-showtext="Show More..." data-target="showMoreContent1" class="btn-expand-more">...Show More</span>
</div>
</div>
</div>

<script src="project.js"></script>
</body>
</html>
```

#########
CSS Coding
#########

```
.card{
width:400px;
margin:10px;
background-color:white;
border:solid 1px;
box-shadow:0 2px 2px 0 rgba(0,0,0,0.1)
}

.placeholder-for-image{
width:400px;
height:250px;
```

```
background-color:red;
}

.expandMoreContent{
height:10px;
overflow:hidden;
transition:height 0.5s ease-in-out;
position:relative;
}

.expandMoreHolder{
padding:15px 0;
text-align:center;
}
.btn-expand-more{
cursor:pointer;
border:1px solid rgba(0,0,0,0.2);
display:inline-block;
}

########
JS Coding
########

const expandsMore=document.querySelectorAll('[expand-more]')

function expand(){
const showContent=document.getElementById(this.dataset.target)
if(showContent.classList.contains('expand-active')){
this.innerHTML=this.dataset.showtext;
}else{
this.innerHTML=this.dataset.hidetext
}
showContent.classList.toggle('expand-active')
}
expandsMore.forEach(expandMore=>expandMore.addEventListener('click',expand)})
}
```

//

95. Anonymous functions

****

```
//anonymous function=Function w/o name
//Often not accessible after its initial creation

//Benefits=Doesn't clutter global namspace
//usable once
//Can be passed as arguments

//IIFE = a JS function that runs as soon as it is defined
//Grouping operator = ()

(function(){
document.getElementById("myH1").innerHTML="Hey!";})();

//setTimeout(function,millisecounds);

let task - function(){document.getElementById("myH1").innerHTML="sup";}

//setTimeout(task,1000);

setTimeout(function(){
document.getElementById("myH1").innerHTML="sup!";}1000);
```

//

96. Closure

******

```
//closure=A function with preserved data.
//Give you access to an outer function's scope from an inner function

//State of variables in outer scope are "saved"
//variables in outer scope are considered "private"

//pretend we are playing a game

let score=function(){

let points=0;
return function(){
points +=1;
return points;
}
}()
console.log(score());
console.log(score());
console.log(score());
```

//

97. Closure examples

****

```
//closure examples w/arguments

function makeSize(size){
return function(){
document.body.style.fontSize=size+"px"
}
}

let size20=makeSize(20);
let size30=makeSize(30);
let size40=makeSize(40);

size20();

function makeFont(font){
return function(){
document.body.style.fontFamily=font;
}
}

let fontInkFree=makeFont("Goudy Stout");
let fontConsolas=makeFont)"consolas");

size(20();
size30();
size40();

fontInkFree();
fontConsolas();
```

//