Javascript Basics

What is it and why?

- A dynamic, interpreted language originally built for web browsers
- Pros:
  - Supported in nearly every browser: Your projects will be public and easily shareable
  - Fast iteration time

  - It's fast
  - There are an enormous number of Javascript developers and resources
  - It's evolving rapidly
- Cons:
  - Weird, confusing behavior (but really, only if you're doing weird things)
  - No type checking (unless you use Typescript)

  - It's slow
  - There are an enormous number of Javascript developers and resources
  - It's evolving rapidly

A Brief Introduction

https://cis700-procedural-graphics.github.io/resources/javascript-basics/

Outline

- Strings and numbers
- How to create objects and properties
- How to define functions
- How to implement object-like behavior
- Callback functions and function pointers
- Modules, imports, and exports

Three.js

Example Three.js Scene

```
const THREE = require('three');
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000 );   // fov, aspect ratio, near, far
camera.position.set(1, 1, 2);                                                  // position the camera
camera.lookAt(new THREE.Vector3(0,0,0));
var renderer = new THREE.WebGLRenderer( { antialias: true } );                 // create a renderer with antialiasing
renderer.setPixelRatio(window.devicePixelRatio);                               // set pixel ratio
renderer.setSize(window.innerWidth, window.innerHeight);                       // set size
renderer.setClearColor(0x020202, 0);                                           // set background color
document.body.appendChild(renderer.domElement);                                // add a canvas to the HTML document

var box = new THREE.BoxGeometry(1, 1, 1);                                      // create a box geometry
var lambert = new THREE.MeshLambertMaterial({ color: 0xeeeeee });              // create a white lambert material
var lambertCube = new THREE.Mesh(box, lambert);                                // create a mesh pointing to the box and lambert
var directionalLight = new THREE.DirectionalLight( 0xffffff, 1 );              // create light
directionalLight.color.setHSL(0.1, 1, 0.95);                                   // set light color
directionalLight.position.set(10, 30, 20);                                     // set light position

scene.add(lambertCube);                                                        // add the cube and light to the scene
scene.add(directionalLight);

renderer.render(scene, camera);                                                // render to the canvas from the view of the camera
```

Animation Draw Loop

```
(function tick() {
   update();                          // perform any updates to the scene
   renderer.render(scene, camera);    // render the scene
   requestAnimationFrame(tick);       // register to call this again when the browser is ready
})();
```

```
var tick = function() {
   update();                          // perform any updates to the scene
   renderer.render(scene, camera);    // render the scene
   requestAnimationFrame(tick);       // register to call this again when the browser is ready
};
tick();
```

Handling Window Resizes

```
window.addEventListener('resize', function() {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
});
```

Getting Started with Project 1

Framework Imports / Exports

```
const THREE = require('three');
const OrbitControls = require('three-orbit-controls')(THREE)
import Stats from 'stats-js'
import DAT from 'dat-gui'


function init(callback, update) {
  // code...
}

export default {
  init: init
}
```

- Here, we import Three.js and a module to handle mouse-camera controls.
- Stats is a module to handle timing how long frame updates take
- Dat.gui is a module for easily adding a GUI to modify variables

- Here we export the object { init: init }
  This allows us to do
  import ModuleName from ‘path/to/Module’
  ModuleName.init(arg1, arg2);

Framework Callback Initialization

```
function init(callback, update) {
  // code...
}
```

Init takes two arguments. The first a function to call when initialization is done. The second is a function to call to perform frame updates.

```
function onLoad(...) {

}

function onUpdate(...) {

}
```

← Given two functions onLoad and onUpdate...

```
init(onLoad, onUpdate);
```

← We can pass them into the init function like this

Framework Initialization

```
function init(callback, update) {
 var stats = new Stats();
 stats.setMode(1);
 stats.domElement.style.position = 'absolute';
 stats.domElement.style.left = '0px';
 stats.domElement.style.top = '0px';
 document.body.appendChild(stats.domElement);


 var gui = new DAT.GUI();


 var framework = {
   gui: gui,
   stats: stats
 };
```

Creation and positioning of the element to show fps info

Create an object “framework” which contains the gui and stats. We will pass this to the “callback” and “update” functions so that they have access to them

Framework Initialization

```
// run this function after the window loads
window.addEventListener('load', function() {
  var scene = new THREE.Scene();
  var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000 );
  var renderer = new THREE.WebGLRenderer( { antialias: true } );
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setSize(window.innerWidth, window.innerHeight);
  renderer.setClearColor(0x020202, 0);

  var controls = new OrbitControls(camera, renderer.domElement);
  controls.enableDamping = true;
  controls.enableZoom = true;
  controls.target.set(0, 0, 0);
  controls.rotateSpeed = 0.3;
  controls.zoomSpeed = 1.0;
  controls.panSpeed = 2.0;

  document.body.appendChild(renderer.domElement);
```

When the window loads, initialize the Three.js scene, camera, render, and camera controls.

The function here is a callback passed to addEventListener because this needs to occur AFTER the browser loads the page. If it is not done this way, there would be no document to append the canvas to

Framework Initialization

```
// resize the canvas when the window changes
window.addEventListener('resize', function() {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
});

// assign THREE.js objects to the object we will return
framework.scene = scene;
framework.camera = camera;
framework.renderer = renderer;
```

Register a callback function to update the camera and renderer on window size updates

Add the scene, camera, and renderer to the framework object

Framework Initialization

```
    // begin the animation loop
    (function tick() {
      stats.begin();
      update(framework); // perform any requested updates
      renderer.render(scene, camera); // render the scene
      stats.end();
      requestAnimationFrame(tick); // register to call this again when the browser renders a new frame
    })();


    // we will pass the scene, gui, renderer, camera, etc... to the callback function
    return callback(framework);
  });
}
```

However often the browser wants (~60fps), call the “update” callback and rerender the scene

Call the callback function now that initialization is complete