

Introduction to Flash Final Project

Create a Flash-based website that incorporates images, sound, video, dynamic text, preloader and components. **Remember to check your code for any errors. It must be letter perfect.** [See sample final project here](#). You will find files to download within these instructions.

In this project you will be working with the following files:

Base movie- adds sound controls and a pre-loader

Main section- adds input and dynamic text

Section one- adds dynamic text which uses the load.data command to import a separate text file. It uses ActionScript to control scrolling.

Section two- implements a photo gallery through the use of arrays

Section three- uses the FLVPlayback component to load video files (.flv)

Design Assessment Rubric- 5 points (1 point each except Assets)

Typography and Navigation: Should be consistent and easy to read.

Layout: Should follow a consistent pattern and appear seamless.

Hyperlinks: ALL should function correctly.

Assets: All assets except video should be original (2 points)

Technical assessment- 15 points

Technical aspects need to have the same functionality seen in the demonstration project.

Base movie: Pre-loader and sound controls (3 points) *Note: Main movie changes are minimal.*

Section One: Dynamic text data loaded from external file (3 points)

Section Two: Photo Gallery (images and titles) (3 points)

Section Three: Video Gallery (3 points)

No submissions will be accepted past the due date.

Getting Started

Download all necessary project files here. Use your computer's utility program or a free 30-day trial of [Stuffit](#) (Win) or [Stuffit](#) (Mac) to open the files. Do not change the names of the folders or move the files from their sub-folders. Examine the directory structure. All the fla files are in the main folder.

When you publish your movies you will create **swf** files for each movie published. The base movie that you rename (see next page) will also produce an **html** file. At the conclusion of this project you will repack all files EXCEPT the fla files and upload them to the dropbox. Now let's get started!

Base Movie

This is the container for additional movies that lay on top of the base movie, like layers in Photoshop. The Base movie determines the stage size, color, and frame rate of ALL movies that will load into it.

1. Rename base_movie.fla as **yourname_fp.fla** and open it in Flash
2. Examine the movie structure (unlock the locked layers)
3. On the main timeline, double click on the section movie clip (nav_mc). Examine the timeline and pre-written ActionScript inside this movie clip. Review the comments so you understand what the code is doing (similar to Lesson 10 of the textbook).
4. Create a replacement background for mcBackground which is on the background layer of the main timeline. You will copy and replace the existing background in all movies with your own background. Make sure in all movies except the base movie it is replacing the content of the guide layer.

Preloader

Animated preloaders give the user precise information about the progress of the loading process and communicates graphically what percentage of the site/page has loaded. The preloader is the first element someone will see when visiting your site. I have created a very simple pre-loader that you can simply drag and drop on to the first frame of your base movie.

1. It is already in your base movie library.
2. Select frame one of the actions layer and write the code written below. **Note: It must be exact.** Do not choose Test Movie yet because you have not built the main.swf

```
1  stop();
2
3  var myLoader:Loader = new Loader();
4  myLoader.contentLoaderInfo.addEventListener(Event.OPEN, showPreloader);
5  myLoader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS, showProgress);
6  myLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, showContent);
7  myLoader.load(new URLRequest("main.swf"));
8
9  var myPreloader:Preloader = new Preloader();
10
11 function showPreloader(event:Event):void {
12     addChild(myPreloader);
13     myPreloader.x = stage.stageWidth / 2;
14     myPreloader.y = stage.stageHeight / 2;
15 }
16
17 function showProgress(event:ProgressEvent):void {
18     var pctLoaded:Number = event.bytesLoaded / event.bytesTotal;
19     myPreloader.loading_txt.text = "Loading - " + Math.round(pctLoaded * 100) + "%";
20     myPreloader.bar_mc.width = 198 * pctLoaded;
21 }
22
23 function showContent(event:Event):void {
24     removeChild(myPreloader);
25     addChild(myLoader);
26 }
```

Explanation of the code: Line 1 stops the main timeline. Line 3-7 creates a variable called myLoader and uses various **contentLoaderInfo** properties of the **Loader** class to monitor the progress of the all the content being loaded into your base movie. Line 9 creates a new class "Preloader" that extends the **MovieClip** class and gives it all the properties and methods of the MovieClip. Line 11-15 creates the function which adds the preloader object, and positions it on the stage. Line 17-21 creates a function that uses properties of the **Loader** class to determine how much data has been loaded. Line 23-26 creates a function that removes the preloader when all data has been loaded, and uses **addChild** to add myLoader to the display list. and loads main.swf (*which we will work with soon*)

Base Movie- Enhancing the home button

Right click on the actions layer, select **Actions**, and add the code below.

```

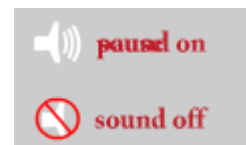
28 function home(event:MouseEvent):void
29 {
30     myLoader.load(new URLRequest("main.swf"));
31     nav_mc.gotoAndStop(nav_mc.start);
32     nav_mc.section_one.enabled = true;
33     nav_mc.section_two.enabled = true;
34     nav_mc.section_three.enabled = true;
35 }
36
37 home_btn.addEventListener(MouseEvent.CLICK, home);
38
39 function contact(event:MouseEvent):void
40 {
41     navigateToURL(new URLRequest("mailto:jdiamond@santarosa.edu"));
42 }
43
44 contact_btn.addEventListener(MouseEvent.CLICK, contact);

```

Explanation of the code: This code assumes your mcNav symbol on the main timeline instance name is nav_mc. Line 28-35 function return nav_mc to the starting state, thereby restoring the nav button colors, and restoring the "enabled" state of all the nav buttons. It also reloads the **main.swf** file which is the default file that sits on top of the base movie. Line 39-42 function launches an email program when someone clicks on the **Contact Us** button. *Make sure to substitute your email address for mine on line 41.*

Base Movie- Creating sound buttons and Loading Sound

1. Once you have rebuilt your background, lock the background layer
2. Select the **audio controls** layer
3. From the assets.fla Library select soundOff, soundOn, soundPause and drag them above the nav_mc movie clip.
4. Give them instance names stop_btn, play_btn, and pause_btn.
5. Position the soundPause button directly over soundOn so that it looks like the image to the right.
6. Find an mp3 file that is no more than 30 seconds long and save it into your **sounds** folder.
7. Right click on the actions layer, select **Actions**, and add the code below. *Note: You will substitute the name of the mp3 file (james_bond.mp3) on line 46 for the mp3 file that you choose to use.*



Explanation for the code: Line 46 creates the **URL Request** object needed to go to a web page. Line 47 creates the **sound** object needed when you load a sound from an external file. Line 48 creates the **sound channel** object which is also required when loading external sounds. Line 49 creates a variable which will hold a numeric value indicating the position of the playback head when the loaded sound is playing. Line 50 loads the soundReq variable (the actual sound from the URL Request) into the new sound variable (sound). Line 51 says that when the sound is completely loaded, run the onComplete function. Line 52 says that when the sound has completely played, run the doneSound function. Lines 54-60 create the doneSound function which is triggered when the sound finishes playing. It makes the pause button invisible, disables it, makes the play button visible and enables it. Lines 62-66 create the onComplete function which is triggered when the sound completely loads. It makes the play and stop button ready to run the playSound and stopSound functions respectively. Lines 68-76 create the playSound function which constructs the soundControl variable. This variable plays the sound and stores the numeric value of the playback head. It then runs the playSound function which makes the pause button visible and enabled, makes the play button invisible and disabled and has the soundControl variable load the doneSound variable when the sound is finished playing. Lines 88-96 creates the stopSound function. It stops the sound and then makes the play button visible and enabled and makes the pause button invisible and disabled. It resets the numeric value of the playhead to 0. Line 97 makes the pause button invisible. This is its default setting since this line is not inside a function and thus does not need to be triggered.

```

46 var soundReq:URLRequest = new URLRequest("sounds/james_bond_test.mp3");
47 var sound:Sound = new Sound();
48 var soundControl:SoundChannel = new SoundChannel();
49 var resumeTime:Number = 0;
50 sound.load(soundReq);
51 sound.addEventListener(Event.COMPLETE, onComplete);
52 soundControl.addEventListener(Event.SOUND_COMPLETE, doneSound);
53
54 function doneSound(event:Event)
55 {
56     pause_btn.visible = false;
57     pause_btn.removeEventListener(MouseEvent.CLICK, pauseSound);
58     play_btn.visible = true;
59     play_btn.addEventListener(MouseEvent.CLICK, playSound);
60 }
61
62 function onComplete(event:Event):void
63 {
64     play_btn.addEventListener(MouseEvent.CLICK, playSound);
65     stop_btn.addEventListener(MouseEvent.CLICK, stopSound);
66 }
67
68 function playSound(event:MouseEvent):void
69 {
70     soundControl = sound.play(resumeTime);
71     pause_btn.visible = true;
72     pause_btn.addEventListener(MouseEvent.CLICK, pauseSound);
73     play_btn.visible = false;
74     play_btn.removeEventListener(MouseEvent.CLICK, playSound);
75     soundControl.addEventListener(Event.SOUND_COMPLETE, doneSound);
76 }
77
78 function pauseSound(event:MouseEvent):void
79 {
80     resumeTime = soundControl.position;
81     soundControl.stop();
82     play_btn.visible = true;
83     play_btn.addEventListener(MouseEvent.CLICK, playSound);
84     pause_btn.visible = false;
85     pause_btn.removeEventListener(MouseEvent.CLICK, pauseSound);
86 }
87
88 function stopSound(event:MouseEvent):void
89 {
90     soundControl.stop();
91     play_btn.visible = true;
92     play_btn.addEventListener(MouseEvent.CLICK, playSound);
93     pause_btn.visible = false;
94     pause_btn.removeEventListener(MouseEvent.CLICK, pauseSound);
95     resumeTime = 0;
96 }
97 pause_btn.visible = false;

```

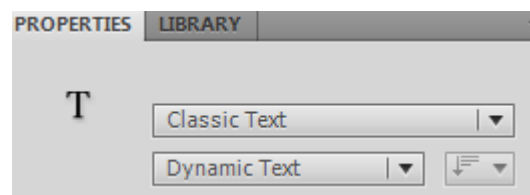
Main Movie

1. Open main.fla
2. Notice that the background layer is a Guide layer. It is only there to make sure that you do not place your content in such a way that it will overlap content from the base movie.
3. Replace the contents of the background layer in the main movie with your background. Make sure to build each layer so your content does not overlap with the background guide layer.
4. Create new banner for the banner/logo layer
5. Provide a description of the content the user will see in the sections of your final project on frame one of the content layer then **save and publish main.fla**

Section_One Movie

This movie uses properties to format dynamically loaded text and ActionScript to make the text scroll continuously on press. We should be able to do this with the UIScrollBar component, this component does not fully function in web browsers. After considerable research I found this excellent solution.

1. Open section_one.fla
2. Copy your new background from your section_one movie and replace the contents of the background layer in the main movie with your background. You want to make sure to build each layer so your content does not overlap with the guide layer.
3. Create new banner and logo for the banner/logo layer
4. Open up **actionscript.txt** from your text sub-folder and answer the questions below then save the file in the same location. *Do not change the tags without knowing what you are doing.*
 - a. What is a variable?
 - b. What is the purpose of a function in ActionScript?
 - c. You have a movie clip instance on the stage with its own self-contained animation. What do you need to assign to that movie clip before you can control it with ActionScript?
 - d. Give an example of how you would tell a specific button to use an event handler you've created when it's clicked?
 - e. Describe the advantages of storing movie clips in the library in Flash CS5.
5. Select the content layer.
6. Select the Text tool, go to the Properties panel and choose Classic Text, Dynamic Text
7. Click and drag a box on the stage to create a dynamic text field.
8. In the Property panel give it the instance name **myTxt**
9. Click on the **Embed** button in the Character dropdown of the text properties panel and embed Uppercase, Lowercase, and punctuation characters.
10. In the paragraph dropdown of the Properties panel, select Set the Multiline behavior
11. Select Window> Common Libraries> Buttons> and drag two instances of the "circle button - next" from the Circle Buttons sub-folder of the classic buttons folder within this Library to the stage.
12. Use the **transform tool** to rotate one button facing up and one button facing down. See picture to the right----->



13. In the Properties panel give an instance name **scrollUp** to the button pointing up and **scrollDown** to the button pointing down.
14. Right-click on frame one of the actions layer, select Actions, and copy and paste what is written below into the Actions panel. *Note you can cut and paste this section of code but check your code for any extra characters.*

```
//-----loading external text and display it-----
var loader:URLLoader = new URLLoader();
var myRequest:URLRequest = new URLRequest("text/actionscript.txt");
loader.load(myRequest);
loader.addEventListener(Event.COMPLETE, textBox);
function textBox(event:Event):void {
    myTxt.htmlText = event.target.data;
    myTxt.background = true;
    myTxt.backgroundColor = 0xCCCCCC;
    myTxt.border = true;
    myTxt.borderColor = 0xC01D29;
    myTxt.wordWrap = true;
}
//-----make buttons to scroll the loaded text-----
var scrolling:Boolean = false;
var scrollDirection:String;

scrollDown.addEventListener(MouseEvent.CLICK,scrollTextDown);
scrollUp.addEventListener(MouseEvent.CLICK,scrollTextUp);
this.addEventListener(MouseEvent.CLICK,stopScroll);

function scrollTextDown(event:MouseEvent):void {
    scrolling = true;
    scrollDirection = "down";
    myTxt.scrollV +=1;
    this.addEventListener(Event.ENTER_FRAME,checkButtons);
}
function scrollTextUp(event:MouseEvent):void {
    scrolling = true;
    scrollDirection = "up";
    myTxt.scrollV -=1;
    this.addEventListener(Event.ENTER_FRAME,checkButtons);
}
function stopScroll(event:MouseEvent):void {
    this.removeEventListener(Event.ENTER_FRAME,checkButtons);
    scrolling = false;
}
//-----checking to see if buttons are pressed-----
function checkButtons(event:Event):void {
    if (scrolling) {
        if (scrollDirection == "down") {
            myTxt.scrollV +=1;
        } else if (scrollDirection == "up") {
            myTxt.scrollV -=1;
        }
    }
}
}
```

Now save and publish section_one fla

Explanation of the code: The code for scrolling the text is broken down into three parts; loading and displaying the external text, making the buttons scroll the text that is loaded, and checking if our buttons are held down.

In the first section we create a **URLRequest** object and pass the path to our txt file. This object knows where our file is. Next we create an **URLLoader** object that loads our txt file. We add an **Event.COMPLETE** to our textLoader, which will fire when our external file is completely loaded. When this happens, we assign the loaded text to the myTxt dynamic text field. This process is handled by the **textBox** function.

In the "make buttons scroll text" section we first declare two variables, one for keeping track if the text is scrolling (scrolling variable, this is of type Boolean, true or false) and the other for checking in which direction the text is scrolling (up or down). Then we add a **MOUSE_DOWN** event to our buttons and a **MOUSE_UP** event to the stage.

The scrollTextDown function handles (tell us what should happen when the left button of our mouse is down) the **MOUSE_DOWN** event added to scrollDown button. Inside this function we set our scrolling variable to true (because we are scrolling the text), set the scrollDirection variable to "down" (we are scrolling down) and increase the **scrollV** property of our text_field with 1. We also add an **ENTER_FRAME** event to the stage.


The scrollTextUp function makes the same thing like scrollTextDown function but for the scrollUp button.

The stopScroll function handles the **MOUSE_UP** event added to the stage. When this event happens we remove the **ENTER_FRAME** event and setting the scrolling variable to false.

In the third section, "checking if our buttons are pressed and hold down" we have the function that handles the **ENTER_FRAME**(fires every frame of our application) event added to the stage. Inside the checkButtons function we use a nested (one inside other) if conditional statement. First we want to know if our text is scrolling (if (scrolling), this is identical with if (scrolling ==true)) and if that is true we want to know in which direction. If is scrolling down we add 1 to the scrollV property of the text_field and if is scrolling up we subtract 1 from scrollV property.

Section_two Movie

This movie uses arrays combined with the **UILoader** component to add dynamic images and dynamic text fields to add dynamic text.

1. Create a set of 6 full-sized and thumbnail-sized images to be used in this section movie.
2. Save the images with an image-editing program like Photoshop to the following dimensions:
 - a. Thumbnails - 50 pixels wide x 77 pixels high at a resolution of 72 ppi
 - b. Full-sized images- maximum of 300 x 300 at a resolution of 72 ppi
3. Name your full-size images **image1.jpg** though **image6.jpg** and place them in the **images** sub-folder of your final_project folder
4. Open section_two fla
5. Copy your new background from your base movie and replace the contents of the background layer in the section_two movie with your background. You want to make sure to build each layer so your content does not overlap with the guide layer.
6. Create new banner for the banner/logo layer
7. Select File> Import to Library and import your thumbnails to the **thumbnail images** folder of the library
8. In the Library double-click on thumb_btn and drag your first thumbnail image onto the image layer of the thumb_btn timeline
9. Repeat this same process with thumb_btn2- 6
10. Select the content layer and drag each of the thumb_btns on to the stage
11. Give them instance names btn1 through btn6
12. Select Window> Components and drag a copy of the **UILoader** onto the stage
13. Name it my_ldr and place a value in the “source” parameter (the Components Parameter drop-down menu in the Properties panel). 
14. Select the text tool
15. Create a dynamic text field set to multi-line, embed characters and name it myTxt.
16. Add some text introducing your photo gallery in this field.
17. Select frame one of the actions layer and write the following, **making sure to substitute your image names and explanations in “myArray” and “desc”**.
18. **Save and publish section_two fla**

```

1  stop();
2
3  var myArray:Array = ["images/image1.jpg", "images/image2.jpg", "images/image3.jpg",
4  "images/image4.jpg", "images/image5.jpg", "images/image6.jpg" ]
5
6  var desc:Array = ["Charlie Brown Farmer", "Charlie Brown Ocean", "Charlie Brown
7  Patissiere", "Charlie Brown Beans", "Charlie Brown Golfer", "Charlie Brown Nature" ]
8
9  btn1.addEventListener(MouseEvent.CLICK, ldr1)
10 function ldr1(event:Event){
11     my_ldr.source = myArray[0];
12     myTxt.text = desc[0];
13 }
14 btn2.addEventListener(MouseEvent.CLICK, ldr2)
15 function ldr2(event:Event){
16     my_ldr.source = myArray[1];
17     myTxt.text = desc[1];
18 }
19 btn3.addEventListener(MouseEvent.CLICK, ldr3)
20 function ldr3(event:Event){
21     my_ldr.source = myArray[2];
22     myTxt.text = desc[2];
23 }
24 btn4.addEventListener(MouseEvent.CLICK, ldr4)
25 function ldr4(event:Event){
26     my_ldr.source = myArray[3];
27     myTxt.text = desc[3];
28 }
29 btn5.addEventListener(MouseEvent.CLICK, ldr5)
30 function ldr5(event:Event){
31     my_ldr.source = myArray[4];
32     myTxt.text = desc[4];
33 }
34 btn6.addEventListener(MouseEvent.CLICK, ldr6)
35 function ldr6(event:Event){
36     my_ldr.source = myArray[5];
37     myTxt.text = desc[5];
38 }

```

Explanation of the code: Line 3 creates an array called **myArray** (like a folder in a file cabinet) that points to files in the images sub-folder of this project. Line 5 creates an array call **desc** that provide textual descriptions of each image. Line 7 makes sure **btn1** listens for a mouse click and when it is clicked runs the function **ldr1**. Line 8-11 creates a function that points to the instance name of the UI Component and loads the first item into the myArray array. Then it loads the text into the dynamic text field which has the instance name myTxt. Lines 12-36 do the same thing for all the other images and text.

Section_three Movie

This movie focuses on loading external video files and downloading the video progressively (so the viewer can watch it as it is downloading). We utilize the **FLV playback** component which lets you control which video plays, whether the video plays automatically, and other aspects of playback. This section assumes you use the flv files located in the videos sub-folder of the downloaded files.

*Note: If you choose to use your own videos and convert them to flv files you will earn **4 extra points** as this is an additional learning process that you would be taking on your own.*

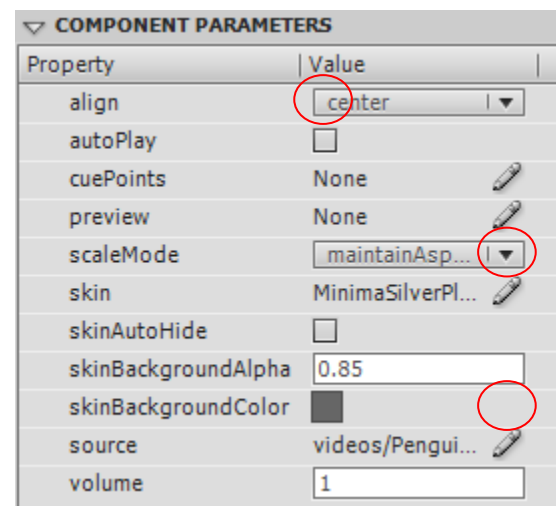
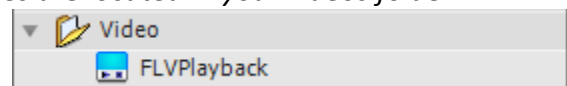
You'll turn autoplay off for the initial penguin movie, so that it doesn't begin playing until the viewer presses the Play button. Then you'll change the **FLVPlayback** component options at different frames to play the appropriate movies for each button, and turn autoplay on, so that the movies start playing immediately when the button is pressed.

1. Open section_three.fla
2. Copy your new background from your base movie and replace the contents of the background layer in the section_three movie with your background. You want to make sure to build each layer so your content does not overlap with the guide layer.
- 3.
4. Customize your individualized banner

Building the Video Player Layer

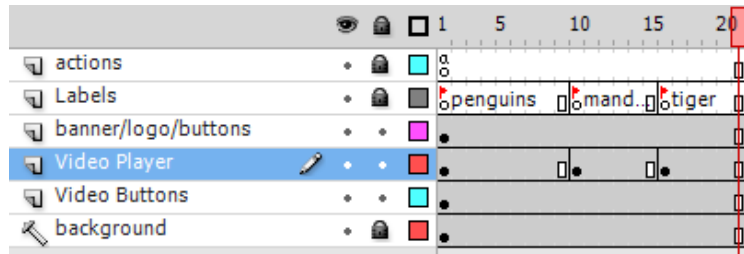
These instructions assume that flv files with the proper names are located in your videos folder.

1. With frame one of the Video Player layer selected choose Window> Components and drag the FLVPlayback component from the Video group on to frame one of the Video Player layer. Size it to your liking.
2. Select the Parameters tab.
3. Deselect the checkbox to the right of autoPlay in the Component Parameters of the Property panel.
4. Choose a skin that you like.
5. Set the source to videos/Penguins.flv
6. Select the Video Player layer and add keyframes on frames 10 and 16.
7. Select the FLVPlayer instance on frame 10, and change the source to videos/Mandrill.flv
8. Select the FLVPlayer instance on frame 16, and change the source to videos/Tiger.flv



Building the Video Buttons Layer

1. Open the Library panel and drag video_button1, video_button2, and video_button3 on to frame one of the Video Buttons layer
2. With all three selected use the Alignment tab to align their bottom edges
3. Select the buttons and give them **instance names** video_button1, video_button2, and video_button3 respectively.
4. Your timeline is complete and should look like this:



Scripting Video Buttons

A different video file is associated with each section of the Timeline

(Penguins.flv with frame 1, Mandrill.flv with frame 10, and so on). You will write the ActionScript that moves the playhead to the appropriate frame in the timeline when the button is clicked. You use the same set of code for each button, with minor changes. The first line of code adds an **event listener**, which listens for the button click, and then runs the function (clickListener1, clickListener2, etc). The function moves the playhead to the appropriate frame in the timeline. When the playhead moves, the appropriate video file plays, according to the controls in the **FLVPlayback** component.

1. Select frame 1 in the Actions layer, and open the Actions panel.
2. Add an **event listener** for the first button by typing the following line in the Actions panel as below. Then copy and paste the lines you just added two times, and then modify each function
3. Your code should look like this:

```
stop();
this.video_button1.addEventListener(MouseEvent.CLICK,clickListener1);
function clickListener1(event:MouseEvent):void {
    gotoAndStop("penguins");
}
this.video_button2.addEventListener(MouseEvent.CLICK,clickListener2);
function clickListener2(event:MouseEvent):void {
    gotoAndStop("mandrill");
}
this.video_button3.addEventListener(MouseEvent.CLICK,clickListener3);
function clickListener3(event:MouseEvent):void {
    gotoAndStop("tiger");
}
```

4. Save and publish section_three.fla

Testing your Final Project

Congratulations. You are now ready to test your Final Project. Make sure that you have both saved and published all your movies. Open up your base movie and select **Control> Test Movie**. If you have error messages bring them in to the lab or come to my online office hours for assistance. Use a program utility like the 30-day free trial of [Stuffit](#) to **compress all .swf files and one html file as well as your images, sounds, text, and videos folders into a single package and [upload the files to the dropbox](#)**. (using your CATE username and password). *Package name: Your name Intro to Flash Final Project*. I hope this has been a valuable learning experience for you on your way to become a serious *Flasher*.