Lab 01: *HTML* Frontend

Interactive Story

HTML

Requires: Chrome Browser

Table of Content: *Implementing an Interactive Story (HTML)*

Lab Introduction

Prerequisites

None. Must have Chrome browser, a code editor, and a github account (free)

Motivation

Understand how to author a multi-page HTML website using much of the HTML syntax.

Goal

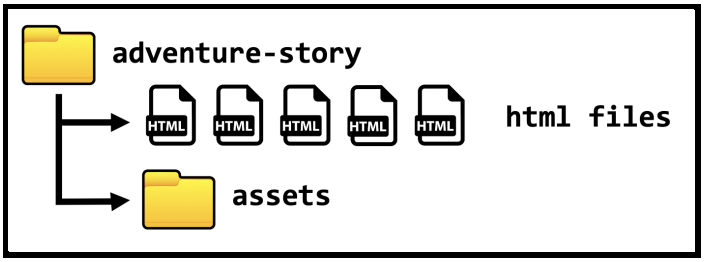
Build an Interactive Story where readers choose their own path.

Learning Objectives

- Structure of an HTML file
- HTML Elements & Attributes
- HTML Inputs with Embedded JavaScript
- Block-level vs Inline-level Elements
- HTML Script tags with Embedded JavaScript
- URL Search Parameters/Query Strings
- Deploy a static web site on github

Project Architecture:

Start this project by downloading the starter files from github.
Create all necessary files and folders as illustrated below.



Download Starter files:
https://github.com/scalemailted/html-adventure-story/archive/refs/heads/master.zip

Concepts

Hyper Text Markup Language (HTML)

HTML is the standard markup language for creating Web pages. HTML describes the structure of a Web page. HTML consists of a series of elements. Defines the app's view, i..e what renders in the web client's view port. Declares the imports for all dependencies such as styles (css) and logic (js)

- **Elements:** An HTML element is defined by a start tag, some content, and an end tag. HTML elements tell the browser how to display the content
- **Attributes:** HTML attributes provide additional information about HTML elements. Attributes are declared within the start tag
- **Block vs Inline:** A block element always starts on a new line and takes up the full width. An inline element uses the existing line and only takes up as much width as necessary.
- **Structure of HTML file:** HTML elements are typically nested within other HTML elements like a tree data structure. The root element is the HTML element. The two children of the root are the head element and the body element
    - **Head:** a container for metadata (data about the HTML document). Metadata is not displayed. Metadata typically defines the document title, character set, styles, scripts.
    - **Body:** defines the document's body. It contains all the contents of an HTML document meant to be viewed such as headings, paragraphs, images, hyperlinks, tables, lists, etc.

Web Client & Web Browsers (Frontend App)

A web browser is an application that parses and displays HTML documents. Web browsers may open HTML documents using HTTP or File protocols

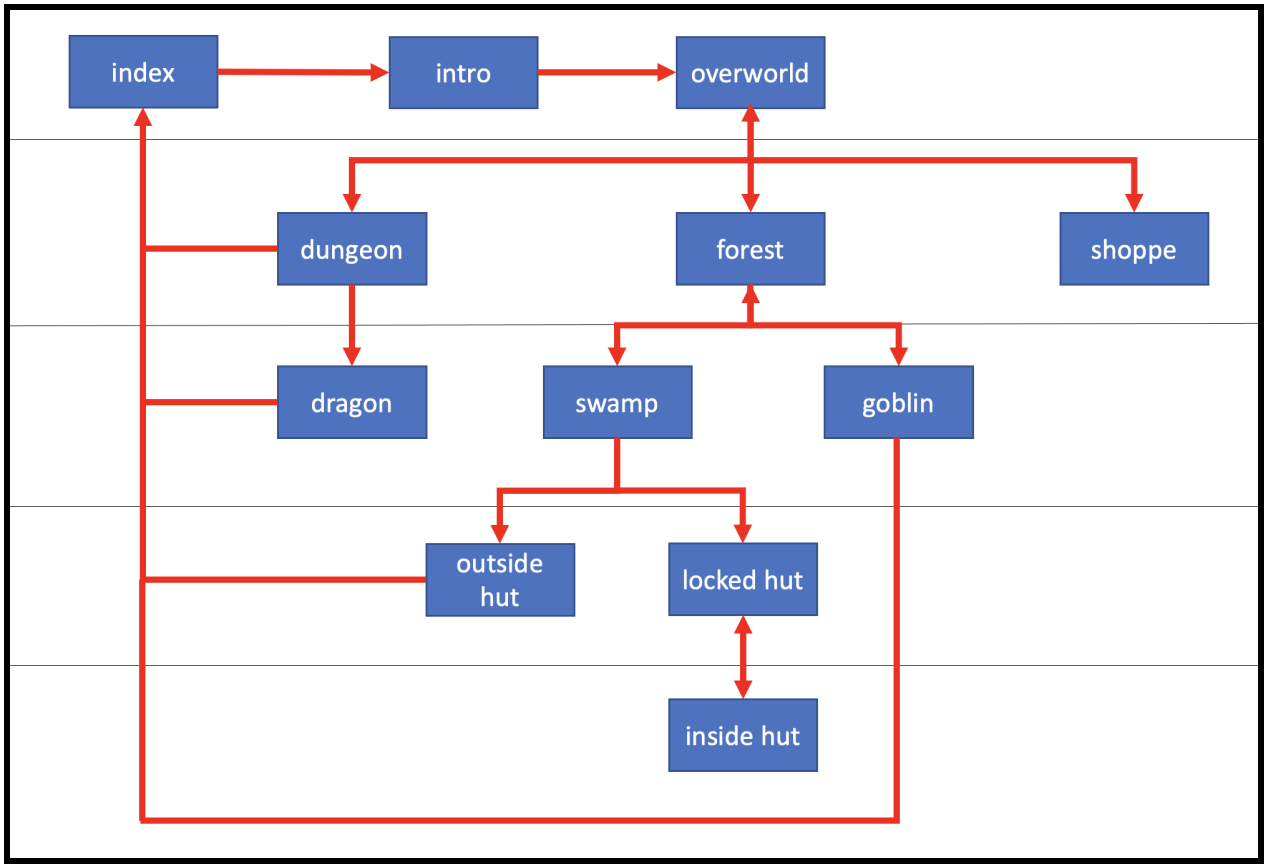
- **HTTP:** HyperText Transfer Protocol is a protocol for transmitting HTML. It's designed for communication between web browsers and web servers.
- **File**: file protocol designed for accessing & reading local files in the browser. The file protocol is more limited than HTTP in browsers for security reasons.
- **HTML Viewer**: Web browsers display the contents of an HTML document in its view port
- **JavaScript Runtime Environment:** Web browsers also have a JSRE for each document loaded into memory. The HTML elements are saved in memory, and accessible by the JSRE

Interactive Story

An interactive story allows the reader to decide which path to take over the course of the story.
Examples: https://itch.io/games/html5/tag-interactive-fiction

Goal 0: Design an Interactive Story

Step 0: Design an Interactive Story
Start with a flowchart that represents each view (i.e. web page). This lab's completed story is illustrated in the flowchart below.



When designing your own interactive story, each scene should either lead to a branch or a dead end. It's beneficial to have multiple branching paths to converge into similar paths. Start by writing each story branch at a time.

Goal 1: Index page

Approach - Design Phase

Our first goal is to define the initial index page where the player starts. All scenes in this interactive story uses HTML in order to define text, images, buttons, hyperlinks, video, audio, etc. The browser tab will include a favicon and title

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

index.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

index.html → <head>

```
<head>
   <title> Adventure Game </title>
   <link rel="icon" href="assets/dragon-logo.png" />
</head>
```

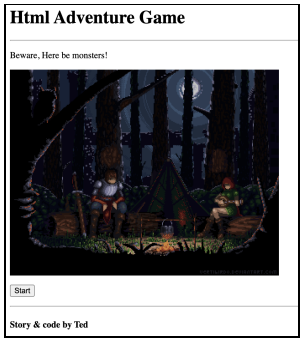
Step 3: Body. HTML elements - Headings, Paragraph, Images, Anchors, Buttons

index.html → <body>

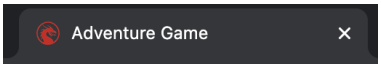
```
<body>
   <h1>Html Adventure Game</h1>
   <hr>
   <p>Beware, Here be monsters!</p>
   <img src='assets/game-title.gif' width="480">
   <p>
       <a href="intro.html">
           <button> Start </button>
       </a>
   </p>
   <hr>
   <h4> Story & Code by Ted </h4>
</body>
```

Assess - Testing Phase

Open the **`index.html`** in a Chrome browser. The tab & the viewport should display text and images.



← browser viewport browser tab→



Goal 2: Intro page

Approach - Design Phase

Our second goal is to introduce the story where it starts. The intro page will be its own HTML document with headings, paragraphs, and hyperlinks. This the target HTML from the index.html page.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

intro.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

intro.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

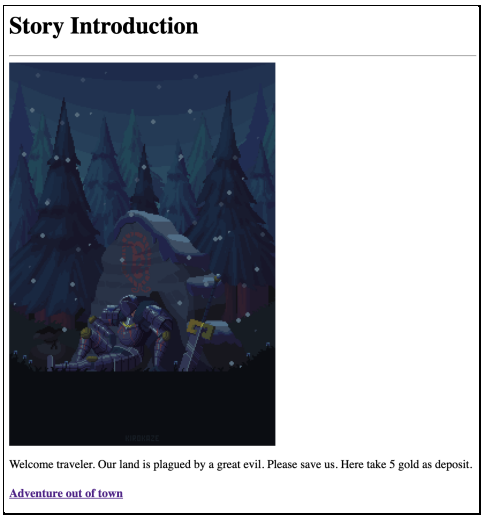
Step 3: Body. HTML elements - Headings, Paragraph, Images, Anchors, Buttons

intro.html → <body>

```
<h1> Story Introduction </h1>
<hr>
<img src='assets/intro.gif' width="360">
<p> Welcome traveler. Our land is plagued by a great evil. Please save us. Here take 5 gold as a deposit. </p>
<h4>
    <a href='overworld.html'>Adventure out of town</a>
</h4>
```

Assess - Testing Phase

Open the **`intro.html`** in a Chrome browser. The viewport should display the following content:



← Browser viewport

Goal 3: Inventory page

Approach - Design Phase

This page displays the player's inventory. HTML documents normally can't share data between each other. However, data may be transmitted between web pages using a query string which is embedded at the end of a URL.
Anatomy of a URL: **address** → https//www.domain.com/page?key1=value1&key2=value2 ← **query string**

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

inventory.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Body. **new** Tables, Table Rows, Table Headings, Table Data, Unicode/Emojis, Comments

inventory.html → <body>

```
Inventory: <hr>
<table>
    <tr>
        <th> &#128176; </th>         <!-- emoji: 💰 -->
        <th> &#128156; </th>         <!-- emoji: 💜 -->
        <th> &#128481; </th>         <!-- emoji: 🗡 -->
        <th> &#128273; </th>         <!-- emoji: 🔑 -->
    </tr>
    <tr>
        <td id='gp'>0</td>
        <td id='hp'>0</td>
        <td id='ap'>0</td>
        <td id='keys'>0</td>
    </tr>
</table>
<script> </script>
```

Step 3: Script. URLSearchParams, location search data, HTML ids, getter methods

inventory.html → <body>

```
var queryString = new URLSearchParams(location.search);
gp.innerText = queryString.get("gp");
hp.innerText = queryString.get("hp");
ap.innerText = queryString.get("ap");
keys.innerText = queryString.get("keys");
```

Assess - Testing Phase

Open the **`inventory.html`** in a Chrome browser without and with a query string

inventory.html	inventory.html?gp=5&hp=3&ap=1&keys=0
Inventory: 💰 🔑 💜 🗡	Inventory: 💰 🔑 💜 🗡 5 0 3 1

Goal 4: Overworld page

Approach - Design Phase

The Overworld scene gives the player a list of places to travel to. Use scripting to pass the queryString to all hrefs. Access the href and src attributes using the HTML ids from the script element.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

overworld.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

overworld.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. **new** Unordered List, List Items, IFrame

overworld.html → <body>

```
<h1>Overworld</h1>
<hr>
<img src='assets/overworld-map.gif' width="480" />
<h4>Where do you want to go?</h4>
<ul>
    <li>  <a id='dungeon' href='dungeon.html'> Dungeon </a>  </li>
    <li>  <a id='forest'  href='forest.html'> Forest   </a>  </li>
    <li>  <a id='shoppe'  href='shoppe.html'> Shoppe   </a>  </li>
</ul>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script>  </script>
```

Step 4: The browser stores the URL in a location variable, which has a property search i.e. the query string.

overworld.html → <script>

```
inventory.src += location.search;          //concat the search parameter to the iframe's src attribute
shoppe.href   += location.search;          //concat the search parameter to the anchor's href attribute
forest.href   += location.search;          //concat the search parameter to the anchor's href attribute
dungeon.href  += location.search;          //concat the search parameter to the anchor's href attribute
```

Step 5: Refactor the URL in intro.html so that the anchor's href includes a query string

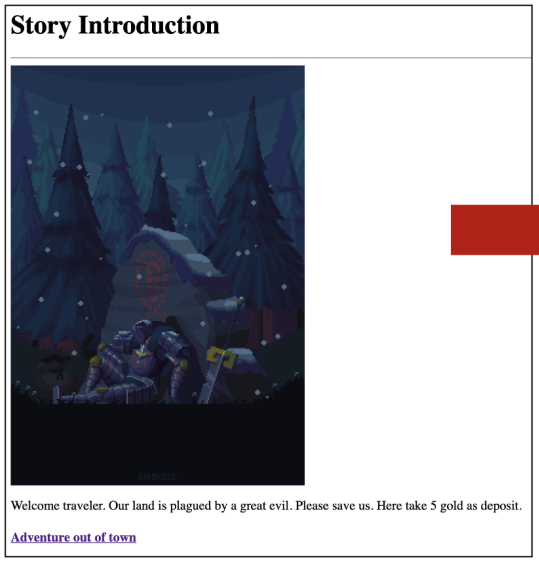
intro.html → <body>

```
<h1> Story Introduction </h1>
<hr>
<img src='assets/intro.gif' width="360">
<p> Welcome traveler. Our land is plagued by a great evil. Please save us. Here take 5 gold as a deposit. </p>
<h4>
    <a href='overworld.html?gp=5&hp=3&ap=1&keys=0'>Adventure out of town</a>
</h4>
```

Assess - Testing Phase

Open the **`intro.html`** in a Chrome browser and click its hyperlink to Adventure out of town. The inventory page should display in the iframe with the appropriate values. Try changing the values and refreshing the page.

intro.html



overworld.html?gp=5&hp=3&ap=1&keys=0

Goal 5: Forest page

Approach - Design Phase

The forest page links from the overworld page. It provides the player with new choices to explore in the story.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

forest.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

forest.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. Story for this scene along with options

forest.html → <body>

```
<h1>Forest</h1>
<hr>
<img src='assets/forest.gif' width='480'>
<p> You wander into the forest. To the left is a winding path up the hills, to the right is a path into the swamps </p>
<h4>What do you do?</h4>
<ul>
    <li> <a id='hills' href='goblin.html'> Go left into the Hills  </a> </li>
    <li> <a id='swamp' href='swamp.html'>  Go right into the Swamp </a> </li>
</ul>
<p>
    <a id='exit' href='overworld.html'>Exit</a>
</p>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Pass state data into new pages via URL query strings

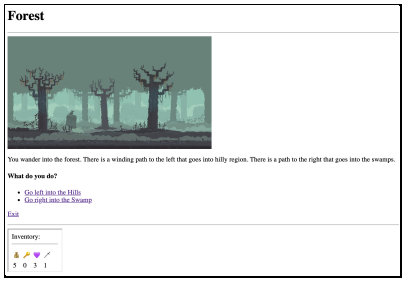
forest.html → <script>

```
inventory.src += location.search;          //append queryString to iframe URL
exit.href     += location.search;          //append queryString to exit href
hills.href    += location.search;          //append queryString to hills href
swamp.href    += location.search;          //append queryString to swamp href
```

Assess - Testing Phase

Open the url: **forest.html?gp=5&hp=3&ap=1&keys=0**

Browser viewport→



Goal 6: Swamp page

Approach - Design Phase

The Swamp scene has an option that results in a game over. To accomplish this, invoke a script function when that option is clicked which overwrites the entire HTML document body with a Game Over scene.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

swamp.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

swamp.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. Story for this scene along with options

swamp.html → <body>

```
<h1>Swamp</h1>
<hr>
<img src='assets/swamp-hut.jpg' width="480">
<p> You come across a creepy witch hut in the swamp. </p>
<h4>What do you want to do?</h4>
<ul>
    <li>  <a id='hut' href='hut.html'> Go into the hut    </a> </li>
    <li>  <a href='#' onclick='die()'> Search outside hut </a> </li>
</ul>
<p>  <a id='exit' href='overworld.html'>Exit</a> </p>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Pass the query to all hrefs & make a function to overwrite the document body with Game Over HTML

swamp.html → <script>

```
inventory.src += location.search;       //append queryString to iframe URL
exit.href     += location.search;       //append queryString to exit href
hut.href      += location.search;       //append queryString to hut href

//function to overwrite the HTML document's body with new HTML elements
function die(){
    document.body.innerHTML = (
        `<h1>You Die</h1>
        <img src="assets/gator.jpg" width="480">
        <p> A giant alligator eats you. </p>
        <a href="index.html"> <h4> Game Over </h4> </a>
        <audio autoplay>
           <source src="assets/scream.mp3">
        </audio>`
    );
}
```

Assess - Testing Phase

Open the **`swamp.html?gp=5&hp=3&ap=1&keys=0`** in a Chrome browser.

swamp.html?gp=5&hp=3&ap=1&keys=0

Swamp

You come across a creepy witch hut in the swamp.

What do you want to do?

- Go into hut
- Search outside the hut

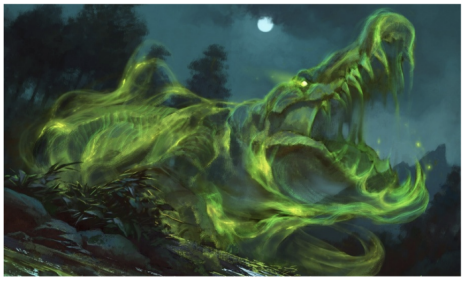
Exit

Inventory:

5 0 3 1

Overwritten HTML triggered from Option 2

You Die



A giant alligator eats you.

Game Over

Goal 7: Hut page

Approach - Design Phase

The Hut scene has a locked door that will not open without a key. With a key, the key breaks, and the door opens, where the player can then decide to click on it to enter into the hut. The door's state may be checked with script logic.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

hut.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

hut.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. Story for this scene along with options

hut.html → <body>

```
<h1>Witch's hut</h1> <hr>
<a id='door' href='#'>
    <img id='doorImage' onclick='openDoor()' src='assets/locked-door.gif' width="240">
</a>
<p id='textbox'></p>
<div> <a id='exit' href='swamp.html'>Exit</a> </div>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Action for the onClick event on the door. Also must pass the QueryString to all hrefs

hut.html → <script>

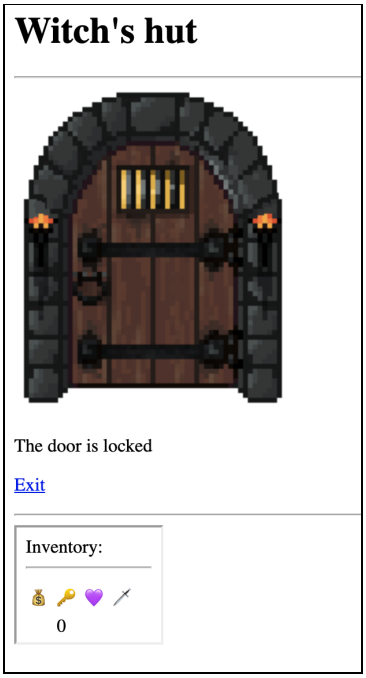
```
inventory.src += location.search;
exit.href     += location.search;

var queryString = new URLSearchParams(location.search);
var key = +queryString.get("keys");

function openDoor(){
    if (doorImage.src.includes('Open-Door')){
        door.href = 'hut-inside.html?'+queryString;
    }
    else if (key > 0){
        doorImage.src='assets/Open-Door.gif'
        textbox.innerHTML = `The door unlocked. but the key broke apart.`
        queryString.set('keys',--key);
        inventory.src = 'inventory.html?'+queryString;
        exit.href = 'swamp.html?'+queryString;
    }
    else {
        textbox.innerHTML = `The door is locked`;
    }
}
```

Assess - Testing Phase

Open the **`hut.html`** in a Chrome browser without a key *(i.e* `keys=0`*)* and with a key *(i.e.* `keys=1`*)*

hut.html?keys=0	hut.html?keys=1
	
After clicking on Door with no key	After clicking on Door with key
	

Goal 8: Hut-Inside page

Approach - Design Phase

The Hut's Inside contains a video clue depicting a sequence of runes that are needed in the dungeon scene. HTML supports both video and audio elements. Options allow videos to autoplay, mute, or loop.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

hut-insidet.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

hut-inside.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. Story for this scene along with options

hut-inside.html → <body>

```
<h1>Witch's Hut</h1>
<hr>
<img src='assets/witchhut-inside.png' width="480" />
<p> You step into the hut and see a magical tome on the desk. You walk toward the book and look at it. </p>
<video autoplay muted loop width=480>
    <source src="assets/Runic-Video.mp4">
</video>
<p> It's opened to a magical spell, and the glyphs glow in sequence. What could it mean? </p>
<div> <a id='exit' href='hut.html'>Exit</a> </div>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script>  </script>
```

Step 4: Script. Concat the query string into all hrefs

hut-inside.html → <script>

```
inventory.src += location.search;
exit.href += location.search;
```

Assess - Testing Phase

Open the **`hut-inside.html`** in a Chrome browser.
The video clue should autoplay and loop

Browser viewport→

Goal 9: Dungeon page

Approach - Design Phase

The dungeon scene uses an HTML input element to get text from the player. If the player enters the correct password they advance to the next scene, otherwise the HTML document is overwritten with a Game Over scene.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

dungeon.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

dungeon.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. **new** Label, Input

dungeon.html → <body>

```
<h1>Dungeon Entrance</h1> <hr>
<p> There is a magic seal preventing you from entering the dungeon. You must dispel. </p>
<img src='assets/magic-seal.gif' width="480">
<p> There's a codex for transliterating runic spells into english. Be careful, incorrect incantations are dangerous!</p>
<img src='assets/rune-translation.png' width="480">
<div>
    <label>Enter the magic word:</label>
    <input type="text" id='inputText'>
    <input type="button" value='Submit' onClick="magicWord()">
</div>
<hr>
<div> <a id='exit' href='overworld.html'>Exit</a> </div>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Function checks value of input & either loads next page or overwrites to Game Over.

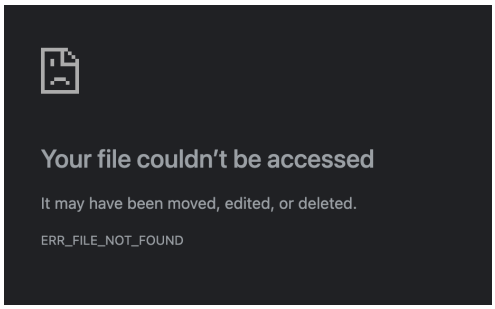
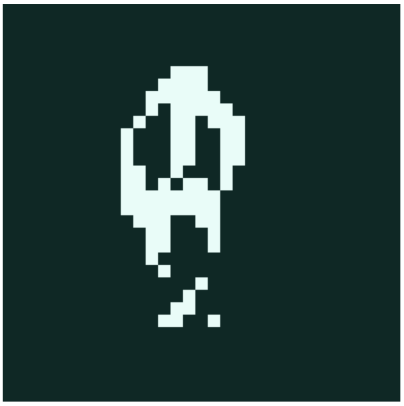
dungeon.html → <script>

```
inventory.src += location.search;
exit.href += location.search;

function magicWord(){
    if (inputText.value == 'open'){
        parent.location='dragon.html'+location.search;
    }
    else {
        document.body.innerHTML = (
            `<h1>You Die</h1>
            <img src="assets/death-spell.gif" width="480">
            <p>You misspoke the magic word and your soul is ripped from your body</p>
            <a href="index.html"> <h4>Game Over</h4> </a>
            <audio autoplay>
                <source src="assets/scream.mp3">
            </audio>`
        );
    }
}
```

Assess - Testing Phase

Open the **`dungeon.html`** in a Chrome browser and try entering the correct word and the wrong word.

`dungeon.html`	`dungeon.html`
Dungeon Entrance There is a magic seal preventing you from entering the dungeon. You must dispell it.  There appears to be a codex for transliterating runic spells into english to speak the magic word. Becareful, incorrect incantations can be dangerous!  Enter the magic word: open Submit Exit Inventory:	**Dungeon Entrance** There is a magic seal preventing you from entering the dungeon. You must dispell it.  There appears to be a codex for transliterating runic spells into english to speak the magic word. Becareful, incorrect incantations can be dangerous!  Enter the magic word: wrong Submit Exit Inventory:
After submitting the correct word: **open**  Your file couldn't be accessed It may have been moved, edited, or deleted. ERR_FILE_NOT_FOUND *Note: the dragon.html file does not exist yet.*	After submitting the incorrect word **You Die**  You mispoke the magic word and your soul is ripped from your body Game Over

Goal 10: Shoppe page

Approach - Design Phase

The Shoppe allows the player to purchase health, attack, or keys. Onclick events trigger script functions to update the values in the Query string. The class URLSearchParams have built-in support for getting & setting values in query.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

shoppe.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

shoppe.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. The HTML elements for the Shoppe scene, onclick attribute triggers script functions

shoppe.html → <body>

```
<h1>Shop</h1> <hr>
<img src='assets/shop.gif' width="480">
<h4>What can I get you?</h4>
<div>
    <button onclick='food()'>   Health &#128156; <br> <i> 1 gp  </i> </button>
    <button onclick='weapon()'> Attack &#128481; <br> <i> 5 gp  </i> </button>
    <button onclick='key()'>    Key &#128273;    <br> <i> 10 gp </i> </button>
</div>
<div>  <a id='exit' href='overworld.html'>Exit</a>  </div>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Functions for all the onClick events. Use URLSerachParams class to get/set the queryString.

shoppe.html → <script>

```
//concat search param to hrefs
inventory.src += location.search;
exit.href += location.search;

//Use URLSerachParams class to get the values
var queryString = new URLSearchParams(location.search);
var gp = +queryString.get("gp");
var hp = +queryString.get("hp");
var ap = +queryString.get("ap");
var keys = +queryString.get("keys");
```

```
function food(){
    if (gp > 0 ){
        queryString.set('gp', gp-1);
        queryString.set('hp', hp+1);
        location.search = queryString;
    }
}

function weapon(){
    if (gp >= 5 ){
        queryString.set('gp', gp-5);
        queryString.set('ap', ap+1);
        location.search = queryString;
    }
}

function key(){;
    if (gp >= 10 ){
        queryString.set('gp',  gp-10);
        queryString.set('keys',keys+1);
        location.search = queryString;
    }
}
```

Assess - Testing Phase

Open the url **`shoppe.html?gp=20&hp=0&ap=0&keys=0`** in a Chrome browser.
Try buying each of the items and verify that the inventory iframe updates both the gold and item counts.

After clicking each button, the item counts should increment by 1 and the gold should reduce.

Goal 11: Goblin page

Approach - Design Phase

The Goblin scene is combat between the player and a goblin. A script function uses the values from the queryString to determine each round in combat. If the player's hp is 0 then game over. If goblin's hp is 0 then player gets gold.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

goblin.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

goblin.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

Step 3: Body. Story for this scene along with options

goblin.html → <body>

```
<h1>Hills</h1>
<hr>
<img id='goblin' src='assets/goblin.gif' width="480">
<p id='battleText'> Goblin ambush! You're threatened by a goblin! </p>
<h4> What do you do? </h4>
<div id='options'>
    <button id='fight' onclick='attack()'> Fight </button>
    <a id='forest' href='forest.html'>
        <button> Retreat </button>
    </a>
</div>
<hr>
<iframe src='inventory.html' id=inventory height=100 width=125></iframe>
<script> </script>
```

Step 4: Script. Action for onClick attack event. and Pass the QueryString to all hrefs

goblin.html → <script>

```
inventory.src += location.search;
forest.href += location.search;

var queryString = new URLSearchParams(location.search);
var gp = +queryString.get("gp");
var hp = +queryString.get("hp");
var ap = +queryString.get("ap");
var goblinHP = 3, goblinAP = 2;
```

```
function attack(){
    goblinHP -= ap;
    if (goblinHP <= 0){
        goblin.src = 'assets/treasure.gif';
        queryString.set('gp',gp+3);
        queryString.set('hp',hp);
        inventory.src = 'inventory.html?'+queryString;
        battleText.innerHTML = `You defeated the goblin and find 3 gold`
        options.innerHTML = `<a href='forest.html?${queryString}'><button>Go back</button></a>`
    }
    else {
        hp -= goblinAP
```

```
        queryString.set('hp',hp);
        inventory.src = 'inventory.html?'+queryString;
        battleText.innerHTML = `You hit for ${ap} damage, it has ${goblinHP}hp left.<br> You take ${goblinAP} damage.`
    }
    if (hp <= 0){
        battleText.innerHTML = `You Died.`
        options.innerHTML = `<a href='index.html'><button>Game Over</button></a>`
    }
}
```

Assess - Testing Phase

Open the **`goblin.html`** in a Chrome browser and try using different hp, ap values to check win/lose conditions

goblin.html?gp=0&hp=10&ap=3	goblin.html?gp=0&hp=1&ap=1
	
	

Goal 12: Dragon page

Approach - Design Phase

The Dragon scene is the final conclusion to the story. It represents the win condition in the story.

Apply - Implementation Phase

Step 1: HTML. Initialize HTML with Head & Body

dragon.html → <html>

```
<html>
    <head></head>
    <body></body>
<html>
```

Step 2: Head. Favicon & Title in Browser tab

dragon.html → <head>

```
<title> Adventure Game </title>
<link rel="icon" href="assets/dragon-logo.png" />
```

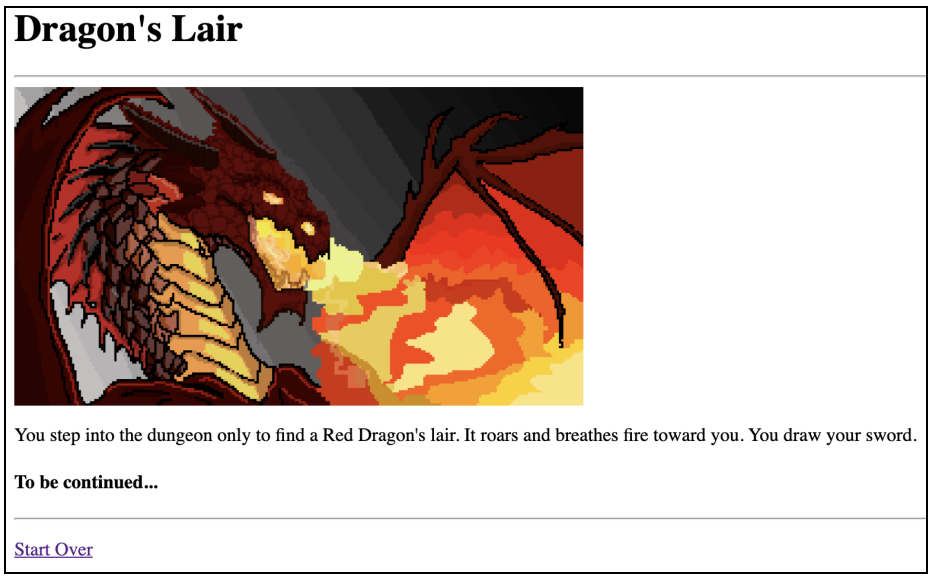
Step 3: Body. Story for this scene along with options

dragon.html → <body>

```
<h1>Dragon's Lair</h1>
<hr>
<img src='assets/dragon-fire.gif' width="480">
<p > You step into the dungeon to find a Red Dragon. It roars and breathes fire toward you. You draw your sword. </p>
<h4>To be continued...</h4>
<hr>
<div>
    <a href='index.html'>Start Over</a>
</div>
```

Assess - Testing Phase

Open the **`dragon.html`** in a Chrome browser.



Conclusions

Final Comments

In this lab you learned to implement a multi-page HTML website. This required defining HTML elements along with HTML attributes. You should understand the difference between block-level elements & inline-level elements. This lab covered many HTML elements such as: html, head, title, link, body, heading, paragraph, anchor, ordered list, unordered list, list items, table, table row, table data, image, video, audio, source, input, select, option, and script.

Future Improvements

- Use CSS to style each HTML page
- Author a more compelling story with better text description, images, audio, video, and options

---

Lab Submission

Compress your project folder into a zip file and submit on Moodle.

Homework 1:

Create your own unique Interactive Story. Use various HTML elements and multiple HTML files. You must have at least 3 branches in your story and persistence between pages with Query Strings.

Homework Bonus:

Showcase bonus. You can receive up to 20 bonus points if your project is outstanding and novel.
I'll publish all showcase projects on UNO's web page as a demo for future students. You should cite such projects on your resume. In order to qualify for the bonus, you must publish a hyperlink of your project into the discord showcase channel.

Appendix

HTML elements

Element	Tag	Type	Description
Heading	`<h1>,<h2>,<h3>, <h4>,<h5>,<h6>`	block	Used for section headings. 1 (largest) to 6 (smallest)
Paragraph	`<p>`	block	Used to wrap text and other HTML elements together
Image	<img>	inline	Used to display images, attributes: ● **`src`**: image's file path, ● **`width`**: image width in browser viewport ● **`height`**: image height in browser viewport
Anchor	`<a>`	inline	Used to link to another HTML file or HTML element,, attributes ● **`href`**: HTML file path or HTML element ID
Button	`<button>`	inline	Used for clickable button

Element	Tag	Type	Description
Code	`<code>`	inline	Used to render code text such as source code, removes original formatting
Pre	`<pre>`	block	Used to render code text & maintain its spaces, tabs, and new lines
Unordered List		block	Used to contain a collection of list items. Denotes each with a bullet point
List Item	`<li>`	block	Used to contain items such as text or other HTML elements for a list

Element	Tag	Type	Description
Table	`<table>`	block	Used to contains table rows, attributes: ● **`border`**: table's border thickness
Table Row	`<tr>`	block	Used to contains table data elements
Table Data	`<td>`	inline	Used to contain text or other html elements
Table Heading	`<th>`	inline	Used to contain text or other html elements but formatted with bold

Element	Tag	Type	Description
Ordered List	`<ol>`	block	Used to contain a collection of list items. Denotes each with a number
Decode - Decimal-based	`&#`	inline	Unicode & emoji characters are encoded as numbers, decimal (base 10) Preface its number with &#
Decode - Hex-based	`&#x`	inline	Unicode & emoji characters are encoded as numbers, hex (base 16) Preface its number with &#x

Element	Tag	Type	Description
Audio	`<audio>`	inline	Used for audio, must contain a source element. Attributes: ● **`controls`**: displays audio player controls ● **`loop`**: repeats audio ● **`autoplay`**: plays audio when page loads, HTTP only)
Source	`<source>`	inline	Used to defines the file path of audio file, Attributes: ● **`src`**: audio file path

Element	Tag	Type	Description
Video	`<video>`	inline	Used for video, must also contain a source element. Attributes: • **`controls`**: displays audio player controls • **`loop`**: repeats audio • **`muted`**: video only, no sound • **`autoplay`**: plays when page loads, (HTTP only unless muted) • **`width`**: video width in viewport

Element	Tag	Type	Description
Division	`<div>`	block	Generic element to contain other HTML elements
Label	`<label>`	inline	Label for an input element
Text Input	`<input>`	inline	A textfield input, Attributes: • **`type="text"`**: sets this input as a textfield • **`id`**: unique identifier to access this element from script, style, html
Button Input	`<input>`	inline	A button input, Attributes: • **`type="button"`**: sets this input as a button • **`value`**: sets text into button • **`onclick`**: sets a JavaScript function to invoke
Source	`<source>`	none	Embeds JavaScript code into HTML. This function checks input's value then selects which HTML page to load

Element	Tag	Type	Description
Break	` `	inline	Breaks a new line
Horizontal rule	`<hr>`	inline	Break a new line and draws a horizontal line
Radio Input	`<input>`	inline	A Radio-type input, Attributes: • **`type="radio"`**: sets this input as a radio buttongroup • **`id`**: unique identifier to access this element from script, style, html • **`name`**: associates all radio elements with same name together • **`value`**: the value for this radio element • **`checked`**: used by javascript to determine if this radio is selected

Element	Tag	Type	Description
Checkbox Input	`<input>`	inline	A Checkbox-type input, Attributes: • **`type="checkbox"`**: sets this input as a checkbox buttongroup • **`id`**: unique identifier to access this element from script, style, html • **`name`**: associates all radio elements with same name together • **`value`**: the value for this radio element • **`checked`**: used by javascript to determine if this radio is selected

Element	Tag	Type	Description
Select	`<select>`	inline	A pulldown input that contains option elements, Attributes: • **`id`**: unique identifier to access this element from script, style, html
Option	`<option>`	inline	Used to set an option in a select input, Attributes: • **`selected`**: sets this option as the default

Emoji Reference: https://www.w3schools.com/charsets/ref_emoji.asp
Window Location: https://www.w3schools.com/js/js_window_location.asp