Comp 125 - Visual Information Processing

Spring Semester 2018 - week 6 - wednesday

Dr Nick Hayward

Semantic HTML - correct usage

- need to ensure elements convey their correct meaning
  - *i.e. the meaning expected for the contained content*
- e.g. often see the following elements mis-used and applied incorrectly for markup,
  - *`<p>` - paragraphs*
  - *`<ul` - unordered list*
  - *`<h1>` to `<h6>` - headings*
  - *`<blockquote>` - blockquote*
- using `<blockquote>` to simply help indent text
  - *instead of CSS margins...*
- or the perennial mis-use of a <p>
  - *simply add extra space between elements*

```
<p> <p>
```

HTML - structure & validation - example

Using lists correctly...

```
<li>nice</li>
<li>cannes</li>
<li>menton</li>
```

- list markup looks OK
  - *still fails validation for an obvious reason*
  - *missing structural grouping for list items*
  - *not valid markup...*
- semantics of the overall list are missing
- example - basic list items

HTML - a semantic point of view

```
<ul>
  <li>nice</li>
  <li>cannes</li>
  <li>menton</li>
</ul>
```

- from the perspective of semantics
  - *meant to act as a group of items that belong together*
- denote such groupings with correct semantic markup
- structuring items to clearly denote their meaning and purpose
- consider global attributes
  - *https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes*
- example - basic group

HTML - benefits of structure & validation

- define and create a meaningful structure for required markup
  - *improves usage and flexibility as project develops*
  - *provides extensible structure for project*
- for example, benefits include
  - *helps increase ease of CSS styling*
  - *creates properly structured documents*
  - *improves general management of updates to markup*
  - ...
- easier to understand and easier to maintain and update
- structured, valid markup aids in repurposing data
  - *into various representations of information*

HTML - benefits of structure & validation - example 1

e.g. a standard list

```
<ul>
  <li>nice</li>
  <li>cannes</li>
  <li>menton</li>
  <li>antibes</li>
  <li>grasse</li>
</ul>
```

- example - basic group style

HTML - benefits of structure & validation - example 2

e.g. lists for navigation, menus, tabs...

```
<ul id="menutabs">
  <li><a href="nice">nice</a></li>
  <li><a href="cannes">cannes</a></li>
  <li><a href="menton">menton</a></li>
  <li><a href="antibes">antibes</a></li>
  <li><a href="grasse">grasse</a></li>
</ul>
```

- example - basic menu tabs

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 4

- use conditional statement to check letter
  - *check against `gameWord` - should return `true` boolean*
  - *check against `answers` - should return `false` boolean*

```
// check letter against game word & not in answers - check for duplicate letter guess
if (gameWord.includes(letter) === true && answers.includes(letter) === false) {
    ...
} else {
    ...
}
```

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 5

- then use `for` loop through `gameWord`
  - *check guess letter against each letter in gameWord*
  - *use loop index `i` to check each value in gameWord*

```
// loop through gameWord
for (i = 0; i < gameWord.length; i++) {
    // check letter against each value in gameWord
  if (gameWord[i] === letter) {
        // add letter to answers array at matching index position
    answers[i] = letter;
    }
}
```

- add guess letter to answers array using loop index `i`

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 6

- also need to keep a record of wrong letter guesses
- use `lettersToGuess` variable
- value is initially set to length of game word

```
// set value for letters to guess from random word
var lettersToGuess = gameWord.length;
```

- then decrement in loop for letter check in `gameWord`

```
lettersToGuess--;
```

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 7

- use `lettersToGuess` to check for end of game
  - *player wins if value reaches 0*

```
// check if gameWord has been guessed correctly
if (lettersToGuess === 0) {
    console.log('game over...player won');
    document.getElementById('guessLetter').innerHTML = 'GAME OVER: word guessed correctly';
    // exit game and reset...need to add
}
```

HTML & JavaScript - create a game - verbose working example

conditional statement and `for` loop

```
// check letter against game word & not in answers - check for duplicate letter guess
if (gameWord.includes(letter) === true && answers.includes(letter) === false) {
  console.log('letter has been found...' + gameWord.includes(letter));
  // loop through gameWord
  for (i = 0; i < gameWord.length; i++) {
    // check letter against each value in gameWord
    if (gameWord[i] === letter) {
      console.log('letter = index ' + i);
      // add letter to answers array at matching index position
      answers[i] = letter;
      // decrement remaining letters to guess to win game...
      lettersToGuess--;
      console.log('letters left to find = ' + lettersToGuess);
      // update game progress to player
      var lettersOutput = answers.join(" "); // create string from answers array
      document.getElementById('wordStatus').innerHTML = 'guess word: ' + lettersOutput;
    }
  }
  // check if gameWord has been guessed correctly
  if (lettersToGuess === 0) {
    console.log('game over...player won');
    document.getElementById('guessLetter').innerHTML = 'GAME OVER: word guessed correctly';
    // exit game and reset...need to add
  }
} else {
  console.log('letter not found...');
  document.getElementById('guessLetter').innerHTML = 'letter not found - please try again...';
  // draw output to hangman...need to add
}
```

References

- W3Schools
  - *JS - conditionals*
  - *JS - For loop*
  - *JS - functions*