Client-Side Web Development

Class 8.1

Today’s Topics

- Template Literals
- Arrays
- Loops
- HTML Creations
- **Exercise:** 99 Luftballons

Announcements

Recording

Web Portfolio

Any Questions?

Template Literals

Template literals **are string that allow for the embedding expressions, string concatenation and multiple lines.**

Template literals **are created by surrounding the contents with a set of backticks (`)**

```
const greeting = `Hello, world.`
```

```
const greeting = `Hello, world.`
```

Backticks

The Template Literal placeholder, `${...}`, is use to insert variables or expressions

```
const subject = 'class'
const greeting = `Hello, ${subject}.`

const calc = `The sum of 3 and 5 is ${3 + 5}.`
```

```
const subject = 'class'
const greeting = `Hello, ${subject}.`

const calc = `The sum of 3 and 5 is ${3 + 5}.`
```

The value of 'subject' is insert into the string

```
const subject = 'class'
const greeting = `Hello, ${subject}.`

const calc = `The sum of 3 and 5 is ${3 + 5}.`
```

greeting = 'Hello, class.'

```
const subject = 'class'
const greeting = `Hello, ${subject}.`

const calc = `The sum of 3 and 5 is ${3 + 5}.`
```

The expression is evaluated first and then inserted into the string

```
const subject = 'class'
const greeting = `Hello, ${subject}.`

const calc = `The sum of 3 and 5 is ${3 + 5}.`
```

calc = 'The sum of 3 and 5 is 8'

8

Arrays

An *array* is an indexed list of values

An *array* can contain values of many different types

An *array* is created surrounding a comma separated list with a set of square brackets

If no list is provided an *empty* array is created

```
const empty = []

const animals = [
  'cat',
  'dog',
  'mouse'
]
```

```
const empty = []
```

This creates an array with no items

```
const animals = [
  'cat',
  'dog',
  'mouse'
]
```

```
const empty = []

const animals = [
  'cat',
  'dog',
  'mouse'
]
```

This creates an array with 3 items

```
const empty = []

const animals = [
  'cat',
  'dog',
  'mouse'
]
```

Commas are required between items

```
const empty = []

const animals = [
  'cat',
  'dog',
  'mouse'
]
```

The last item should NOT have a comma after it.

```
const empty = []

const animals = [
  'cat',
  'dog',
  'mouse'
]
```

The return between each item is optional

Each item is automatically assigned a numbered index starting with `0`

```
const empty = []

const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]
```

An the value of an item of an array is accessed using the item's index

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[0] // cat
animals[1] // dog
animals[2] // mouse
```

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[0] // cat
animals[1] // dog
animals[2] // mouse
```

Use the variable name with the array

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[0] // cat
animals[1] // dog
animals[2] // mouse
```

Use square brackets around the index

The equals sign (=) can be used to reassign the value of an item

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[2] = 'cow' // ['cat', 'dog', 'cow']
```

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[2] = 'cow' // ['cat', 'dog', 'cow']
```

This index holds the value 'mouse'

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals[2] = 'cow' // ['cat', 'dog', 'cow']
```

The value 'cow' will replace 'mouse'

The `push()` method will add a new item to the end of an array

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals.push('cow') // 3
animals.push('duck') // 4

animals[4] // duck
```

The `join()` method is used to "join" the array items into a string

The `join()` method takes one parameter, the glue, to join the item. The default is a comma.

```
const animals = [
  'cat',  // 0
  'dog',  // 1
  'mouse' // 2
]

animals.join() // cow,dog,mouse

animals.join(' | ') // cow | dog | mouse
```

Loops

Loops **are statements that are used to repeat a block of code until a certain condition is met**

Loops **can also be used to iterate over an array or object**

For Loop

For loops **are used when the number of iterations is known**

For loops **have a expression and a block code that will be executed each iteration**

The expression of a for loop has three parts:
initialization*, *condition*, and *iteration

```
// Log 'I will not 'quack' in class 1000 times
console.log(`1. I will not 'quack' in class`)
console.log(`2. I will not 'quack' in class`)
console.log(`3. I will not 'quack' in class`)
console.log(`4. I will not 'quack' in class`)
console.log(`5. I will not 'quack' in class`)
console.log(`6. I will not 'quack' in class`)
console.log(`7. I will not 'quack' in class`)
...
console.log(`999. I will not 'quack' in class`)
console.log(`1000. I will not 'quack' in class`)
```

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

Initialization

Creates a variable to be used as an iterator

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

Condition

The loop will continue as long as the condition is true

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

Iteration

Increases the value of the iterator by one

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

Must use 'let'

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

This doesn't need to be 'i'

```
// Log 'I will not 'quack' in class 1000 times
for (let i = 1; i <= 1000; i++) {
  console.log(`${i}. I will not 'quack' in class`)
}
```

Required

The *for loop* can be used to iterate over arrays, but better options exist

HTML Creation

One of the most common tasks of JavaScript is to create new HTML elements

There are two common techniques for creating new HTML element with JavaScript

The manual creation of elements using the `createElement()` method

The automatic creation of elements by using a string of HTML and the `innerHTML` property

The `innerHTML` property

The `innerHTML` property is used to get and set a string representing an element's children

When getting an element `innerHTML` it will be returned as a string

```html
<div id="list">
  <h2>Animals</h2>
</div>
```

```
const $list = document.getElementById('list')

$list.innerHTML // <h2>Animals</h2>
```

When setting the value of an element's `innerHTML`, a string of HTML must be provided

When setting the value of an element's `innerHTML`, any existing HTML will be replaced

```
const list = `
<ul>
  <li>Cat</li>
  <li>Dog</li>
  <li>Mouse</li>
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = list
```

```
const list = `
<ul>
  <li>Cat</li>
  <li>Dog</li>
  <li>Mouse</li>
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = list
```

The Result

- cat
- dog
- mouse

```
const list = `
<ul>
  <li>Cat</li>
  <li>Dog</li>
  <li>Mouse</li>
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

```
const list = `
<ul>
  <li>Cat</li>
  <li>Dog</li>
  <li>Mouse</li>
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Insert existing HTML

```
const list = `
<ul>
  <li>Cat</li>
  <li>Dog</li>
  <li>Mouse</li>
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

The Result

Animals

- cat
- dog
- mouse

It possible to easily create repetitive HTML with the use of arrays and loops

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

This array holds an animal for each list item

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

This array will be used to hold each list item as HTML

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Iterated over the animals array

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Creates the list item HTML for each animal

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

The HTML is then added to the items array

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Create the template for an unordered list

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Inserts the list items added to the items array in the for loop

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

Adds the unordered list to the page

```
const animals = ['cat', 'dog', 'mouse', 'cow', 'horse', 'duck']
const items = []

for (let i = 0; i < animals.length; i++) {
  items.push(`<li>${animals[i]}</li>`)
}

const list = `
<ul>
  ${items.join('')}
</ul>`

const $list = document.getElementById('list')
$list.innerHTML = $list.innerHTML + list
```

The Result

Animals

- cat
- dog
- mouse
- cow
- horse
- duck

Examples

Exercise: 99 Luftballons

For next class...

- **Review:** Shake, Rattle and Roll
- **Lab:** Dominoes