Video 2.3
Chris Murphy

SD4x-2

Variables in JavaScript

- Five primitive types: number, string, boolean, null, undefined

- Sometimes we may want to have a collection of ordered values

- Sometimes we may want to have a collection of associated values with semantically meaningful names/keys

Arrays

- Arrays are used to store a list of values in a single variable

- Values can be of any type, and are split with commas and wrapped in square brackets

```
var myArray = ['cars', 12, false];
```

Arrays

- Arrays are used to store a list of values in a single variable

- Values can be of any type, and are split with commas and wrapped in square brackets

- Values can be accessed with $arrayVar[index]$

```
var myArray = ['cars', 12, false];

var age = myArray[1];
console.log(age);            // 12
```

Arrays

- Arrays are used to store a list of values in a single variable

- Values can be of any type, and are split with commas and wrapped in square brackets

- Values can be accessed with *arrayVar[index]*

```
var myArray = ['cars', 12, false];

var age = myArray[1];
console.log(age);              // 12
myArray[2] = true;
console.log(myArray[2]);    // true
```

Arrays

- Arrays are used to store a list of values in a single variable

- Values can be of any type, and are split with commas and wrapped in square brackets

- Values can be accessed with *arrayVar[index]*

- The length of an array can be found with `.length`

```
var myArray = ['cars', 12, false];

var age = myArray[1];
console.log(age);            // 12
myArray[2] = true;
console.log(myArray[2]);    // true

console.log(myArray.length); //3
```

Array Indices

- When **reading** an array value by its index, *arrayVar[index]* will return `undefined` if the index is out of bounds

```
var a = ['cat', 'dog', 'banana'];

console.log(a[4]);  // undefined

console.log(a[-9]); // undefined
```

Array Indices

- When **reading** an array value by its index, *arrayVar[index]* will return `undefined` if the index is out of bounds

```
var a = ['cat', 'dog', 'banana'];

console.log(a[4]);  // undefined

console.log(a[-9]); // undefined
```

Array Indices

- When **reading** an array value by its index, *arrayVar[index]* will return `undefined` if the index is out of bounds

```
var a = ['cat', 'dog', 'banana'];

console.log(a[4]);  // undefined

console.log(a[-9]); // undefined
```

Array Indices

- When **reading** an array value by its index, *arrayVar[index]* will return `undefined` if the index is out of bounds

```
var a = ['cat', 'dog', 'banana'];

console.log(a[4]);  // undefined

console.log(a[-9]); // undefined
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

  - add an element at that index if
    ```
    index >= arrayVar.length
    ```

  - create a mapping from the index to the element if
    ```
    index < 0
    ```

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined x 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

  - add an element at that index if
    `index >= arrayVar.length`

  - create a mapping from the index to the element if
    `index < 0`

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined x 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index,
  arrayVar[index] will

  - add an element at that index if
    ```
    index >= arrayVar.length
    ```

  - create a mapping from the index to the element if
    ```
    index < 0
    ```

```javascript
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined × 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index,
  arrayVar[index] will

  - add an element at that index if
    `index >= arrayVar.length`

  - create a mapping from the index to the element if
    `index < 0`

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined × 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

    - add an element at that index if
      `index >= arrayVar.length`

    - create a mapping from the index to the element if
      `index < 0`

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined × 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

  - add an element at that index if
    `index >= arrayVar.length`

  - create a mapping from the index to the element if
    `index < 0`

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined × 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

  - add an element at that index if
    ```
    index >= arrayVar.length
    ```

  - create a mapping from the index to the element if
    ```
    index < 0
    ```

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined x 1, "panda", -5: "elephant"]
```

Array Indices

- When **writing** an array value by its index, *arrayVar[index]* will

  - add an element at that index if
    `index >= arrayVar.length`

  - create a mapping from the index to the element if
    `index < 0`

```
var a = ['cat', 'dog', 'banana'];

a[4] = 'panda';
console.log(a[4]);   // "panda"
console.log(a[3]);   // undefined

a[-5] = 'elephant';
console.log(a[-5]);  // "elephant"

console.log(a);
// (5) ["cat", "dog", "banana", undefined × 1, "panda", -5: "elephant"]
```

Adding to an Array

- Elements can be added to arrays using `push()` and `unshift()`

  - `push()` will add elements to the end of the array

  - `unshift()` will add elements to the beginning of the array

```
var myArray = ['car', 'bike'];

myArray.push('scooter');
console.log(myArray);              // car,bike,scooter

myArray.unshift('train');
console.log(myArray);              // train,car,bike,scooter
```

Adding to an Array

- Elements can be added to arrays using `push()` and `unshift()`

  - `push()` will add elements to the end of the array

  - `unshift()` will add elements to the beginning of the array

```
var myArray = ['car', 'bike'];

myArray.push('scooter');
console.log(myArray);              // car,bike,scooter

myArray.unshift('train');
console.log(myArray);              // train,car,bike,scooter
```

Adding to an Array

- Elements can be added to arrays using `push()` and `unshift()`

  - `push()` will add elements to the end of the array

  - `unshift()` will add elements to the beginning of the array

```
var myArray = ['car', 'bike'];

myArray.push('scooter');
console.log(myArray);          // car,bike,scooter

myArray.unshift('train');
console.log(myArray);          // train,car,bike,scooter
```

Adding to an Array

- Elements can be added to arrays using **`push()`** and **`unshift()`**

  - **`push()`** will add elements to the end of the array

  - **`unshift()`** will add elements to the beginning of the array

```
var myArray = ['car', 'bike'];

myArray.push('scooter');
console.log(myArray);          // car,bike,scooter

myArray.unshift('train');
console.log(myArray);          // train,car,bike,scooter
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);              // scooter
console.log(myArray);              // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);              // train
console.log(myArray);              // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);              // scooter
console.log(myArray);              // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);              // train
console.log(myArray);              // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);               // scooter
console.log(myArray);               // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);               // train
console.log(myArray);               // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);                // scooter
console.log(myArray);                // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);                // train
console.log(myArray);                // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);              // scooter
console.log(myArray);              // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);              // train
console.log(myArray);              // car,bike
```

Removing from an Array

- Elements can be removed from arrays using `pop()` and `shift()`

  - `pop()` will remove and return an element from the end of the array

  - `shift()` will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);              // scooter
console.log(myArray);              // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);              // train
console.log(myArray);              // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);             // scooter
console.log(myArray);             // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);             // train
console.log(myArray);             // car,bike
```

Removing from an Array

- Elements can be removed from arrays using **pop()** and **shift()**

  - **pop()** will remove and return an element from the end of the array

  - **shift()** will remove and return an element from the beginning

```
var myArray = ['train', 'car', 'bike', 'scooter'];

var vehicle = myArray.pop();
console.log(vehicle);              // scooter
console.log(myArray);              // train,car,bike

vehicle = myArray.shift();
console.log(vehicle);              // train
console.log(myArray);              // car,bike
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```javascript
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)    // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
   name: 'John Doe',
   age: 25,
   isMale: true,
   personality: ['patient', 'loyal', 'happy'],
   company: { name: 'edX', id: 2984 }
}

console.log(person.age);            // 25
console.log(person['company'].id)   // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
   name: 'John Doe',
   age: 25,
   isMale: true,
   personality: ['patient', 'loyal', 'happy'],
   company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
   name: 'John Doe',
   age: 25,
   isMale: true,
   personality: ['patient', 'loyal', 'happy'],
   company: { name: 'edX', id: 2984 }
}

console.log(person.age);            // 25
console.log(person['company'].id)   // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);          // 25
console.log(person['company'].id)   // 2984
```

Objects

- JavaScript objects are used to store key-value pairs

- Values can be of any type, including arrays and objects!

- Values can be accessed by *myObject.property* or *myObject['property']*

```
var person = {
    name: 'John Doe',
    age: 25,
    isMale: true,
    personality: ['patient', 'loyal', 'happy'],
    company: { name: 'edX', id: 2984 }
}

console.log(person.age);              // 25
console.log(person['company'].id)     // 2984
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);       // 11

pet['status'] = 'good boy';
console.log(pet.status);    // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);          // undefined
pet.age = 11;
console.log(pet.age);          // 11

pet['status'] = 'good boy';
console.log(pet.status);       // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Modifying Objects

- Key-value pairs can be added to objects, even after their initial declaration

```
var pet = {
    name: 'Cooper',
    type: 'dog'
}

console.log(pet.age);        // undefined
pet.age = 11;
console.log(pet.age);        // 11

pet['status'] = 'good boy';
console.log(pet.status);     // "good boy"
```

Summary

- JavaScript **arrays** let us create ordered collections of values with numeric indices

- JavaScript **objects** are collections of associated values with semantically meaningful names/keys