



Video 2.5

Chris Murphy

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
  var product = 1;  
  for (var i = 1; i <= n; i++) {  
    product *= i;  
  }  
  return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```


Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Declaring and Using Functions

```
function factorial(n) {  
    var product = 1;  
    for (var i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
var x = . . .  
  
var f = factorial(x);  
  
console.log(f);
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

```
function isEven(n) {  
    return n % 2 == 0;  
}  
nums.every(isEven); // true
```


Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

```
function isEven(n) {  
    return n % 2 == 0;  
}  
nums.every(isEven); // true
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

```
function isEven(n) {  
    return n % 2 == 0;  
}  
nums.every(isEven); // true
```

```
function square(n) {  
    return n * n;  
}  
var squares = nums.map(square); // [ 16, 64, 144, 4 ]
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

```
function isEven(n) {  
    return n % 2 == 0;  
}  
nums.every(isEven); // true
```

```
function square(n) {  
    return n * n;  
}  
var squares = nums.map(square); // [ 16, 64, 144, 4 ]
```

Applying Functions to Arrays

```
var nums = [ 4, 8, 12, 2 ];
```

```
function print(n) {  
    console.log(n);  
}  
nums.forEach(print);
```

```
function isEven(n) {  
    return n % 2 == 0;  
}  
nums.every(isEven); // true
```

```
function square(n) {  
    return n * n;  
}  
var squares = nums.map(square); // [ 16, 64, 144, 4 ]
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```


Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

```
function changeMe(obj) {  
    obj.age++;  
}  
var p = { age: 30 };  
changeMe(p);  
console.log(p.age); // now 31
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

```
function changeMe(obj) {  
    obj.age++;  
}  
var p = { age: 30 };  
changeMe(p);  
console.log(p.age); // now 31
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

```
function changeMe(obj) {  
    obj.age++;  
}  
var p = { age: 30 };  
changeMe(p);  
console.log(p.age); // now 31
```

Pass-by-Value vs. Pass-by-Reference

- Primitive arguments are passed by **value**: the function cannot change them

```
function tryToChange(x) {  
    x = 4;  
}  
var y = 11;  
tryToChange(y);  
console.log(y); // still 11
```

- Object arguments are passed by **reference**: the function **can** change them

```
function changeMe(obj) {  
    obj.age++;  
}  
var p = { age: 30 };  
changeMe(p);  
console.log(p.age); // now 31
```

Functions as Objects

- JavaScript functions are objects
 - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

Functions as Objects

- JavaScript functions are objects
 - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

```
var add = function (a, b) {  
    return a + b;  
};  
  
console.log(add(3, 5));           // 8
```


Functions as Objects

- JavaScript functions are objects
 - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

```
var add = function (a, b) {  
    return a + b;  
};  
  
console.log(add(3, 5));           // 8
```

Functions as Objects

- JavaScript functions are objects
 - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

```
var add = function (a, b) {  
    return a + b;  
};  
  
console.log(add(3, 5));           // 8
```

Functions as Objects

- JavaScript functions are objects
 - Therefore, functions can take advantage of the benefits of an object, such as having properties
- Since JavaScript functions are objects, we can have variables refer to them

```
var add = function (a, b) {  
    return a + b;  
};  
  
console.log(add(3, 5));           // 8
```

Functions in Objects

- JavaScript functions can also be declared and used in objects

```
var johnDoe = {  
  name: 'John Doe',  
  age: '32',  
  greeting: function () {  
    return 'Hello! Nice Meeting You!';  
  }  
}  
  
console.log(johnDoe.greeting());
```

Functions in Objects

- JavaScript functions can also be declared and used in objects

```
var johnDoe = {  
  name: 'John Doe',  
  age: '32',  
  greeting: function () {  
    return 'Hello! Nice Meeting You!';  
  }  
}  
  
console.log(johnDoe.greeting());
```

Functions in Objects

- JavaScript functions can also be declared and used in objects

```
var johnDoe = {  
  name: 'John Doe',  
  age: '32',  
  greeting: function () {  
    return 'Hello! Nice Meeting You!';  
  }  
}  
  
console.log(johnDoe.greeting());
```

Functions in Objects

- JavaScript functions can also be declared and used in objects

```
var johnDoe = {  
  name: 'John Doe',  
  age: '32',  
  greeting: function () {  
    return 'Hello! Nice Meeting You!';  
  }  
}  
  
console.log(johnDoe.greeting());
```

Object Prototypes

- Every object in JavaScript has a **prototype**, accessed from the `__proto__` property in the object.
- The `__proto__` property is also an object, with its own `__proto__` property, and so on
- The root prototype of all objects is `Object.prototype`
- An object inherits the properties of its prototype

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
  this.name = name;
  this.age = age;
  this.greeting = function () {
    return 'Hello! My name is ' + this.name;
  }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
    this.name = name;
    this.age = age;
    this.greeting = function () {
        return 'Hello! My name is ' + this.name;
    }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```


Creating a Prototype

- Prototypes are created like any other JavaScript function or object
- The **this** keyword refers to the current object
- The **new** keyword can be used to create new objects from the same prototype

```
function Person (name, age) { // prototype
  this.name = name;
  this.age = age;
  this.greeting = function () {
    return 'Hello! My name is ' + this.name;
  }
}

var johnDoe = new Person('John Doe', 32);
johnDoe.greeting(); // Hello! My name is John Doe

var janeDoe = new Person('Jane Doe', 28);
janeDoe.greeting(); // Hello! My name is Jane Doe
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```

Extending a Prototype

- Prototypes can extend another prototype with more functionality
- To inherit a prototype, set the `__proto__` property of an object to the parent prototype

```
function Student (name, age, school) {  
  this.__proto__ = new Person(name, age);  
  this.school = school;  
}  
  
var sarahBrown = new Student('Sarah Brown', 17, 'PennX');  
  
sarahBrown.greeting();           //Hello! My name is Sarah Brown  
sarahBrown instanceof Person;    //true
```


Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Prototype Properties

- Properties and methods can be added to prototypes by adding them to the prototype property

```
var Person = function (name, age, occupation) {  
    this.name = name;  
    this.age = age;  
    this.occupation = occupation;  
}  
  
Person.prototype.planet = 'Earth';  
Person.prototype.introduction = function () {  
    return 'I am a ' + this.occupation;  
}  
  
var johnDoe = new Person('John Doe', 32, 'Dentist');  
  
johnDoe.planet;           //Earth  
johnDoe.introduction();   //I am a Dentist
```

Summary

- JavaScript supports functions
 - Primitives are passed by value
 - Objects are passed by reference
- Functions are objects and can be used to create objects
- JavaScript prototypes can be used to create “blueprints” for objects and can be modified dynamically