Functions and Scope

- function
- scope
- hoisting

Define a Function

<div class="columns"> <div>

- Function Declaration

```
function name([param[, param[, ... param]]]) {
   [statements]
}
```

- Function Expression

```
var name = function([param[, param[, ... param]]]) {
   [statements]
};
```

</div> <div>

Function Hoisting

Move declarations to the top of the current scope.
<u>Only declarations are hoisted, not initializations.</u>

```
foo(); // foo
fun(); // Error

function foo() {
  console.log('foo');
}

var fun = function () {
  console.log('fun');
};
```

Function Scope & Variable Masking

<p class="mark">scope.js</p>

```
var x = 1;

function foo() {
  var x = 2;
  console.log(x); // 2
}
console.log(x); // 1
```

Function Invocation

<p class="mark">functions.js</p>

- function invocation
- method invocation
    - `this` - context
- constructor invocation
    - If a function or method invocation is preceded by the keyword new, then it is a constructor invocation
- indirect invocation
    - `call` & `apply`

Function Arguments and Parameters

<p class="mark">arguments.js</p>

- optional parameters
- arguments object
    - `arguments` is an array-like object
- **default parameters**
- **rest parameters**

Value vs Reference

<p class="mark">value_reference.js</p>

- Primitive types are passed by value
- Objects are passed by reference

Using Object Properties As Arguments

```
function foo(options) {
  var name = options.name || 'default';
  var age = options.age || 0;
  console.log(name, age);
}

foo({ name: 'John', age: 20 });
foo({ name: 'John' });
```

Functions As Values

- functions are first-class objects
- functions can be assigned to variables
- functions can be passed as arguments to other functions
- functions can be returned from functions

Closure

<p class="mark">closure.js, iife.js</p>

- A closure is a function that has access to the parent scope, even after the parent function has closed.
- IIFE - Immediately Invoked Function Expression

Object Oriented Programming

- Class
  - a blueprint for creating objects
- Object
  - an instance of a class
- Inheritance
  - a class can inherit properties and methods from another class
- Polymorphism
  - a class can override inherited methods

Back to the old days

We had builtin classes like `Date`, `Array`, etc. We can create instances of these classes using the `new` keyword. But what if we want to create our own classes?
<div class="columns">

```
var d1 = new Date();
var d2 = new Date();

var arr1 = new Array(); // arr1 = []
var arr2 = new Array(1, 2); // arr2 = [1,2]
```

```
var person1 = new Person();
var person2 = new Person();
```

Constructor Function

- A constructor function is a function that returns an object
- It is used with the `new` keyword to create an instance of an object
- It is a convention to capitalize the first letter of a constructor function

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

var person = new Person('Aaron', 30);
```

`new` Keyword

- Creates an empty object
- Sets the value of `this` to the new object
- Adds a property called `__proto__` to the new object
- Adds a `return this` to the end of the function

`this` Keyword

<p class="mark">this.js</p>

- `this` is a special keyword that refers to the object that is being created

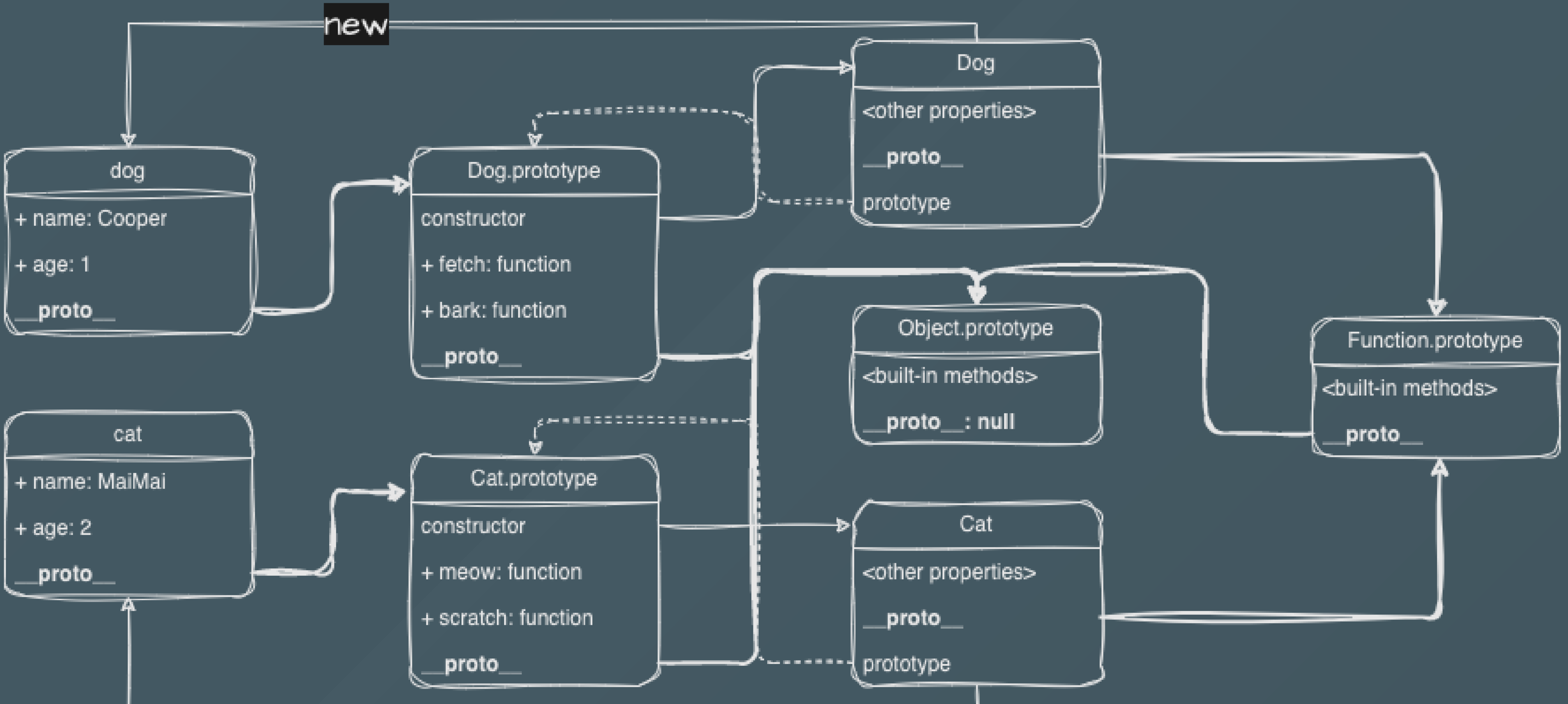
```
function callName() {
  console.log(this.name);
}

const obj = {
  name: 'obj',
  callName: callName
};

callName(); // undefined
obj.callName(); // obj
```

`this` binding

- implicit binding
  - `obj.callName()` - `this` refers to `obj`
- explicit binding
  - `callName.call(obj)` - `this` refers to `obj`
- `new` binding
- lexical binding
  - arrow function
- global binding
  - `global` or `window`, depends on the runtime environment

`__proto__` Property

<p class="mark">proto.js</p>



Class

<p class="mark">class.js</p>

- A class is a blueprint for creating objects
- It is a convention to capitalize the first letter of a class

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

const person = new Person('Aaron', 30);
```

Static Method

- A static method is a method that is called on the class itself, not on an instance of the class

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  static create(name, age) {
    return new Person(name, age);
  }
}
const person = Person.create('Aaron', 30);
```

Public and Private Properties

<p class="mark">class_property.js</p>

- Public properties are accessible outside of the class
- Private properties are only accessible inside of the class

Inheritance

- A class can inherit properties and methods from another class
- The class that is being inherited from is called the parent class or super class

<div class="columns">

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  sayHello() {
    console.log(`Hello, my name is ${this.name}`);
```

`super` Keyword

- `super` is a special keyword that refers to the parent class
- It is used to call the constructor of the parent class
- Use `super` before `this`. This ensures that superclass is initialized before subclass