OOP

POWERED BY:

object-oriented programming

- ➔ create simple shopping cart.
- ➔ add 5 items in it:
  2 x Coca~Cola (2.5 KM)
  1 x Pringles (4 KM)
  2 x Kiki (1 KM)
- ➔ add function to calculate total.

```
class Product {
 constructor(name, value) {
   this.name = name;
   this.value = value;
 }
}

class Basket {
 constructor() {
   this.products = []
 }

 addProduct(product, amount) {
   this.products.push(...Array(amount).fill(product))
 }

 getTotal() {
   return this.products.map(p => p.value).reduce((a, b) => a + b, 0)
 }
}

const cocaCola = new Product('Coca~Cola', 2.5);
const pringles = new Product('Pringles', 4);
const kiki = new Product('Kiki', 1);
const basket = new Basket();

basket.addProduct(cocaCola, 2);
basket.addProduct(pringles, 1);
basket.addProduct(kiki, 2);
console.log(basket.getTotal());
```

constructor

- **function** that is used to create **class** instances.
- should start with capital letter
- ES6 has **class** keyword that is just sugar coated version of **constructor function**

```
function Animal() {}
Animal.prototype.talk = function() {
 console.log(this.sound);
}
const cat = new Animal();

// these two are the same

class Animal {
 talk() {
   console.log(this.sound)
 }
}
const cat = new Animal();
```

__proto__ vs prototype

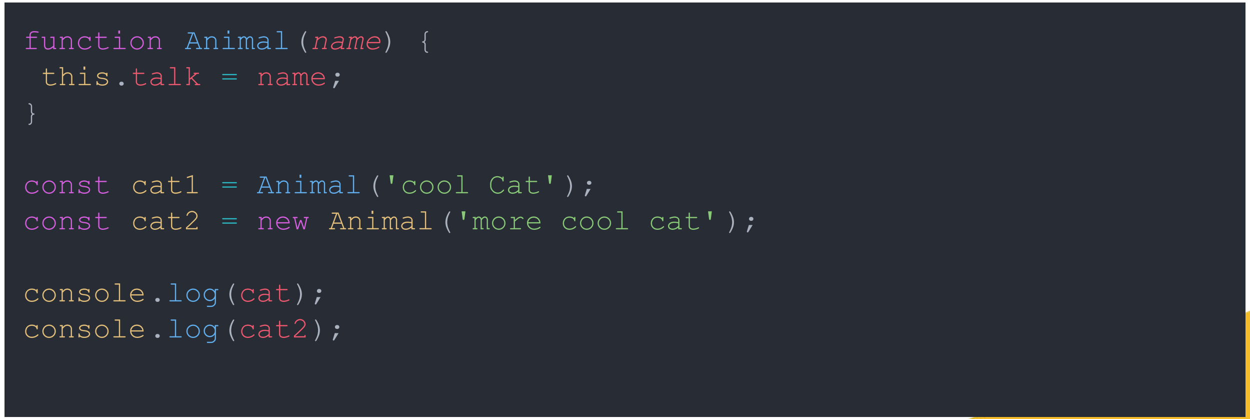
- object has **__proto__**
- function has **prototype**

```
function Animal() {}
Animal.prototype.talk = function() {
 console.log(this.sound);
}

const cat = new Animal();
cat.sound = 'old meow!'
cat.talk();
```

new

- a brand new **object** is created (aka, constructed) out of thin air
- the newly constructed object is prototype-linked
- the newly constructed object is set as the **this** binding for that function call
- unless the function returns its own alternate object, the new-invoked function call will automatically return the newly constructed object.



```
function Animal(name) {
 this.talk = name;
}

const cat1 = Animal('cool Cat');
const cat2 = new Animal('more cool cat');

console.log(cat);
console.log(cat2);
```

this

- **this** is binding that is made when a **function is invoked**, and what it references is determined entirely by the **call-site** (the location in code where a function is called not where it's declared)
- **this** is not a reference to the **function** itself, nor is it a reference to the function's **lexical scope**

```
const cat = {
 sound: 'meow!',
 makeSound: function() {
   console.log(this.sound);
 }
}
cat.makeSound(); // meow!

const makeSound = cat.makeSound;
makeSound(); // ??
```

this

- new binding (**new** keyword)
- explicit binding (**bind**, **call**, **apply**) *(can't be mixed with **new**)*
- implicit binding (object method)
- default binding

```
const bar = new foo()
const bar = foo.call( obj2 )
const bar = obj1.foo()
const bar = foo()
```

bind, call, apply

- explicitly set **this**
- **call** and **apply** invoke function
- **bind** just makes binding, doesn’t invoke the function

```
const cat = {
 sound: 'meow!',
 talk: function() {
   console.log(this.sound);
 }
};

const dog = {
 sound: 'woof!'
};

cat.talk.call(dog); // woof!
dog.talk = cat.talk.bind(dog);
dog.talk(); // woof!
```

```
// the new way
class Animal { // ES6 class
 talk() {
   console.log(this.sound);
 }
}

const cat = new Animal(); // create new instance
cat.sound = 'new meow!'; // add property to the instance
cat.talk(); // call method that lives in Animal

// override the method
Animal.prototype.talk = function() {
 console.log('NO WAY!!');
};
cat.talk();

// extend original class
class EnhancedAnimal extends Animal {
 constructor() {
   super(); // super is Animal
 }
 roll() {
   console.log('rolllling');
 }
}

const enhancedCat = new EnhancedAnimal(); // create new instance
enhancedCat.sound = 'enhanced new meow!'; // add property to the instance
enhancedCat.talk(); // call method that lives in Animal
enhancedCat.roll(); // call method that lives in EnhancedAnimal
```

POWERED BY:

```javascript
// the old way
function Animal() {} // constructor function
Animal.prototype.talk = function() {
 console.log(this.sound);
};

const cat = new Animal(); // create new instance
cat.sound = 'old meow!'; // add property to the instance
cat.talk(); // call method that lives in Animal

// override the method
Animal.prototype.talk = function() {
 console.log('NO WAY!!');
};
cat.talk();

// extend original class
function EnhancedAnimal() {
 Animal.call(this);
}
// this is crucial, with this line we have access to Animal methods
EnhancedAnimal.prototype = Object.create(Animal.prototype);
EnhancedAnimal.prototype.constructor = EnhancedAnimal;
EnhancedAnimal.prototype.roll = function() {
 console.log('rollllling');
};

const enhancedCat = new EnhancedAnimal(); // create new instance
enhancedCat.sound = 'enhanced new meow!'; // add property to the instance
enhancedCat.talk(); // call method that lives in Animal
enhancedCat.roll(); // call method that lives in EnhancedAnimal
```

POWERED BY:

```
// the bad way
const animal = {
 talk: function() {
   console.log(this.sound);
 }
};

const cat = {
 sound: 'meow!'
};

// this is bad for performance
Object.setPrototypeOf(cat, animal); // connect cat to animal
cat.talk();
animal.talk = function() {
 console.log('NO WAY!!');
};
cat.talk();

const enhancedAnimal = {
 roll() {
   console.log('rollllling');
 }
}
Object.setPrototypeOf(enhancedAnimal, animal); // connect enhancedAnimal to animal

const enhancedCat = {
 sound: 'enhanced bad meow!'
}
Object.setPrototypeOf(enhancedCat, enhancedAnimal);
enhancedCat.talk(); // call method that lives in Animal
enhancedCat.roll(); // call method that lives in EnhancedAnimal
```

```
// the best way
const animal = {
 talk: function() {
   console.log(this.sound);
 }
};

const cat = Object.create(animal); // create new object with animal as
prototype
cat.sound = 'best meow!';
cat.talk();

// override default method
animal.talk = function() {
 console.log('NO WAY!!');
};
cat.talk();

const enhancedAnimal = Object.create(animal); // animal as prototype
enhancedAnimal.roll = function() {
 console.log('rollllling');
}
const enhancedCat = Object.create(enhancedAnimal); // enhancedAnimal as
prototype
enhancedCat.sound = 'enhanced best meow!';
enhancedCat.talk(); // call method that lives in Animal
enhancedCat.roll(); // call method that lives in EnhancedAnimal
```

exercise

1. create the **constructor function** for a **Video** object. The function should take in arguments of **title** (a string), **uploader** (a string, the person who uploaded it), and **seconds** (a number, the duration), and it should save them as properties of the object.
2. create a method on the Video protoype called **watch**(). When that method is called, it should use **console.log** to output a string like "You watched all 60 seconds of Otters Holding Hands!"
3. Instantiate a **new Video** object and call the **watch**() method on it.
4. Instantiate another Video object with **different constructor arguments**.
5. bonus: use an array of data to instantiate 5 Video objects.
6. bonus: make the watch method accept **amounts** of seconds to watch for, and call it with different amounts of seconds.

exercise

1. define a **new class** called MusicVideo that **extends** Video. Its **constructor** should also take in an **artist** argument.
2. instantiate a new MusicVideo object and call the **watch**() method on it.
3. add a method to MusicVideo called **rockOut**() that uses console.log to output a string like "You rocked out to La Bamba by ritchie Valens!.
4. bonus: use an array of data to instantiate 5 MusicVideo objects.
5. bonus: make an array of video data with both normal videos and music videos, loop through them, and decide on each one whether to make it a Video or MusicVideo object.

exercise

1. create a class called **Date** that includes three pieces of information as instance variables, month, day, year.
2. class should have a constructor that initializes the three instance variables and check that the values provided are correct.
3. provide a **set** and a **get** method for each instance variable.
4. provide a method **displayDate** that displays the month, day, and year separated by forward slashes (/)

POWERED BY: nsoft sportradar

SPARK. SCHOOL

exercise

imagine a tollbooth at a bridge. cars passing by the booth are expected to pay a 50 cent toll. Mostly they do, but sometimes a car goes by without paying. the tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected.

1. create class called **TollBooth** that has number of cars, and the total amount of money collected. initialize both of these to 0.
2. method **payingCar()** increments the car total and adds 0.50 to the cash total.
3. method **nopayCar()** increments the car total but adds nothing to the cash total.
4. method **display()** displays the two totals

call these functions randomly with promises.

POWERED BY: nsoft sportradar

SPARK. SCHOOL

exercise

1. create class **Heater** that has **temp**, **increment**, **min** and **max**
2. **temp** is 0 and **increment** is 5 by default and you pass **min** and **max** to constructor
3. make get/set for **increment**, **min** and **max**
4. make methods **warmer** and **cooler** that will increment/decrement temperature by **increment** value
5. make sure **temp** can’t go above **max** and below **min**, **increment** can’t be negative

POWERED BY:

exercise

1. create class **Vehicle** with props **hasEngine**, **isElectric**, **canDrive**, **capacity**, **isRunning**, **mileage** and **drive** method that takes in amount of kilometers
2. create **Car** and **Bike** classes that extend **Vehicle** and set **doorNumber** accordingly. add **tireNumber** as props
3. add **carStart**, **carStop**, **bikeStart**, **bikeStop** accordingly
4. instantiate **batMobile**, **batBike** and **flintstonesCar**
5. drive them around (start engine, drive, stop engine)
6. every car should go to **Garage** if it has mileage above 200
7. make **Garage** class that has **queue** (array of cars), **addVehicle**, **removeVehicle, checkVehicles** and **fixVehicle** method that decreases mileage by 20km.

Only vehicles that have mileage below 100 are fixed!!

POWERED BY: nsoft sportradar

SPARK. SCHOOL

THANK YOU

for attention