Code Nation

JavaScript Fundamentals

Objects

{codenation}®

First thing's first

Create a function that logs "try pressing chocolate and then espresso in the coffee machine"

If condition "goodtaste" is true and there's enough money left, output "You're in for a good time!"

Learning Objectives

- To understand the concept of an object
- To access data from within an object
- To use functions with objects
- To understand and use the “this” keyword

{codenation}®

Introducing Objects

objects are containers that can store data and functions.

They're also dead good

We use key-value pairs to store data inside an object

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

Create **variable** called **cafe** that stores an object

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

The {} determines that this is an **object** (not just a simple variable or array)

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

name, **seatingCapacity**, **hasSpecialOffers** and **drinks** are all **keys**

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

keys and **values** are separated by a **colon**.

The values are on the right with each value separated by a comma.

key: value

```
const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ]
};
```

Create **variable** called **cafe** that stores an object

The {} determines that this is an **object** (not just a simple variable or array)

name, **seatingCapacity**, **hasSpecialOffers** and **drinks** are all **keys**

keys and **values** are separated by a **colon**.

The values are on the right with each value separated by a comma.

key: value

Activity:

Let's create an object called **person** with a key called **name** and set the **value** to your name.

Activity:

Let's create an object called **person** with a key called **name** and set the **value** to your name.

Add another **key** called **age**.

Values can be any data type – they can even be arrays, or even functions

So we have data inside objects — we better be able to access that data. Guess what we use?

object.property

person.name

```
console.log(person.name);
```

But that's not all – we can also use bracket notation

```
console.log(person["name"]);
```

```
person.name  vs  person["name"]
```

```
person.name  vs  person["name"]
```

Both common, both worth knowing.

Bracket notation actually gives us a bit more flexibility

You can use variables to select the keys of an object

Which sounds gloriously confusing :o

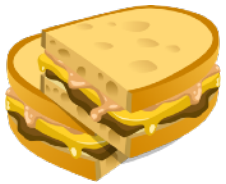
But actually it just means that Whitesheep may well decide to introduce a special offer

and we can use variables to help us implement it

Let's say Whitesheep may have different specials based in the time of the day...

Free croissants at breakfast... 

Free drink with a sandwich at lunch...




```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: [
        "Cappuccino",
        "latte",
        "filter coffee",
        "tea",
        "hot chocolate"
    ],

    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer"
};
```

We could put each special in an object and select one at specific time

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer"
};

if (time < 1100){
    offer = cafe.breakfastOffer;
    console.log(cafe.breakfastOffer);
} else if (time < 1500){
    offer = cafe.lunchOffer;
    console.log(cafe.lunchOffer);
}
```

Activity:

Let's create an alarm.

Create a key called **weekendAlarm**, with a value saying "no alarm needed", and a key called **weekdayAlarm**, with a value saying "get up at 7am"

Create a variable called day and one called alarm

If day is Saturday or Sunday, set alarm to weekendAlarm

If the day is a weekday, set alarm to weekdayAlarm

Adding properties

Objects are mutable, which is a posh way of saying we can still change them once we've made them

To add to our objects, we can use either brackets or dot notation

```
cafe.biscuits = ["waffle", "shortbread"];
```

```
cafe.biscuits = ["waffle", "shortbread"];
```

Or

```
cafe["biscuits"] = ["waffle", "shortbread"];
```

Activity:

Let's add a list of favourite songs to our person object and log them to the console.

Using Functions with Objects

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer”,

     openCafe:()=>{
         return "Come on in";
     },
     closeCafe:()=>{
         return "We are closed, come back tomorrow!"
     }
};

console.log(cafe.openCafe());
console.log(cafe.closeCafe());
```

Since **ES6**, a modern version of JavaScript, it's easier.

You don't need the colon, nor the arrow syntax to create functions inside an object.

```
openCafe:()=>{
    return "Come on in";
},
closeCafe:()=>{
    return "We are closed, come back tomorrow!"
}
```

In ES6:

```
openCafe(){
    return "Come on in";
},
closeCafe(){
    return "We are closed, come back tomorrow!"
}
```

They both work perfectly fine :)

Using methods to operate on data inside functions

So, let's push functions a little further and have them operate on data within our object

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer",
        openCafe(){
            if(hasSpecialOffers){
                return "Time for a special offer!";
            }
        },
    closeCafe(){
        return "We are closed, come back tomorrow!";
    }
};

console.log(cafe.openCafe());
```

!Error! :o

ReferenceError: hasSpecialOffers is not defined

hasSpecialOffer is actually outside of the function's scope

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer",
        openCafe(){
            if(hasSpecialOffers){
                return "Time for a special offer!";
            }
        },
    closeCafe(){
        return "We are closed, come back tomorrow!";
    }
};

console.log(cafe.openCafe());
```

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer",
        openCafe(){
            if(hasSpecialOffers){
                return "Time for a special offer!";
            }
        },
    closeCafe(){
        return "We are closed, come back tomorrow!";
    }
};

console.log(cafe.openCafe());
```

We need to tell openCafe where hasSpecialOffers is

We do that using this keyword

```
let offer = "none";
let time = 1200;

const cafe = {
    name: "Whitesheep",
    seatingCapacity: 100,
    hasSpecialOffers: false,
    drinks: ["Cappuccino","latte","filter coffee","tea","hot chocolate"],
    breakfastOffer: "free croissant with coffee",
    lunchOffer: "free drink with surprisingly priced sandwich",
    none: "Sorry no offer",
        openCafe(){
            if(this.hasSpecialOffers){
                return "Time for a special offer!";
            }
        },
    closeCafe(){
        return "We are closed, come back tomorrow!";
    }
};
console.log(cafe.openCafe());
```

this means this current object

So accessing this.hasSpecialOffer from inside the object is the same as saying cafe.hasSpecialOffers outside it

Activity:

Let's edit our person object to include...

A function called sayHi and when it's called, it should return "Hello, my name is ${this.name}"

Learning Objectives

- To understand the concept of an object
- To access data from within an object
- To use functions with objects
- To understand and use the “this” keyword

{codenation}®

Activity:

Create an object called pet with key values of:

name, typeOfPet, age, colour

And methods called eat and drink. They should return a string saying [Your pet name] us eating/drinking.