WE CAN REPRESENT EVERYDAY STUFF WITH JS OBJECTS

Since common things have “bits” of related info, they make good Object examples

BOOK

title
author
publisher

numChapters
numPages
illustrator

Each property is an important bit of data associated with a book.

WE CAN REPRESENT EVERYDAY STUFF WITH JS OBJECTS

Since common things have “bits” of related info, they make good Object examples

BOX

height	volume
width	material
length	contents

Because an Object contains multiple bits of info, it's often called a "composite value."

AN OBJECT'S PROPERTIES CAN BE ASSIGNED VALUES

Like with everyday objects, properties can point to specific amounts or qualities

BOX

height	: 6	volume	: 480
width	: 8	material	: "cardboard"
length	: 10	contents	: booksArray

Properties can refer to numbers, strings, arrays, functions, and even other Objects!

CREATING AN OBJECT WITH JAVASCRIPT

There are multiple ways to build Objects…let's look first at the "Object literal."

BOX

height	: 6	volume	: 480
width	: 8	material	: "cardboard"
length	: 10	contents	: booksArray

```
var myBox = { };
```

A set of curly brackets says to make a new object…in this case, however, it's an empty one with no properties.

CREATING AN OBJECT WITH JAVASCRIPT

There are multiple ways to build Objects...let's look first at the "Object literal."

BOX

height	: 6	volume	: 480
width	: 8	material	: "cardboard"
length	: 10	contents	: booksArray

```
var myBox = { height: 6 };
```

Adding a property involves creating a name for the property using a string, and then assigning a value to it using a :.

CREATING AN OBJECT WITH JAVASCRIPT

There are multiple ways to build Objects…let's look first at the "Object literal."

BOX

height	: 6	volume	: 480
width	: 8	material	: "cardboard"
length	: 10	contents	: booksArray

```
var myBox = { height: 6, width: 8, length: 10, volume: 480 };
```

Multiple properties are separated by commas.

CREATING AN OBJECT WITH JAVASCRIPT

There are multiple ways to build Objects…let’s look first at the “Object literal.”

BOX

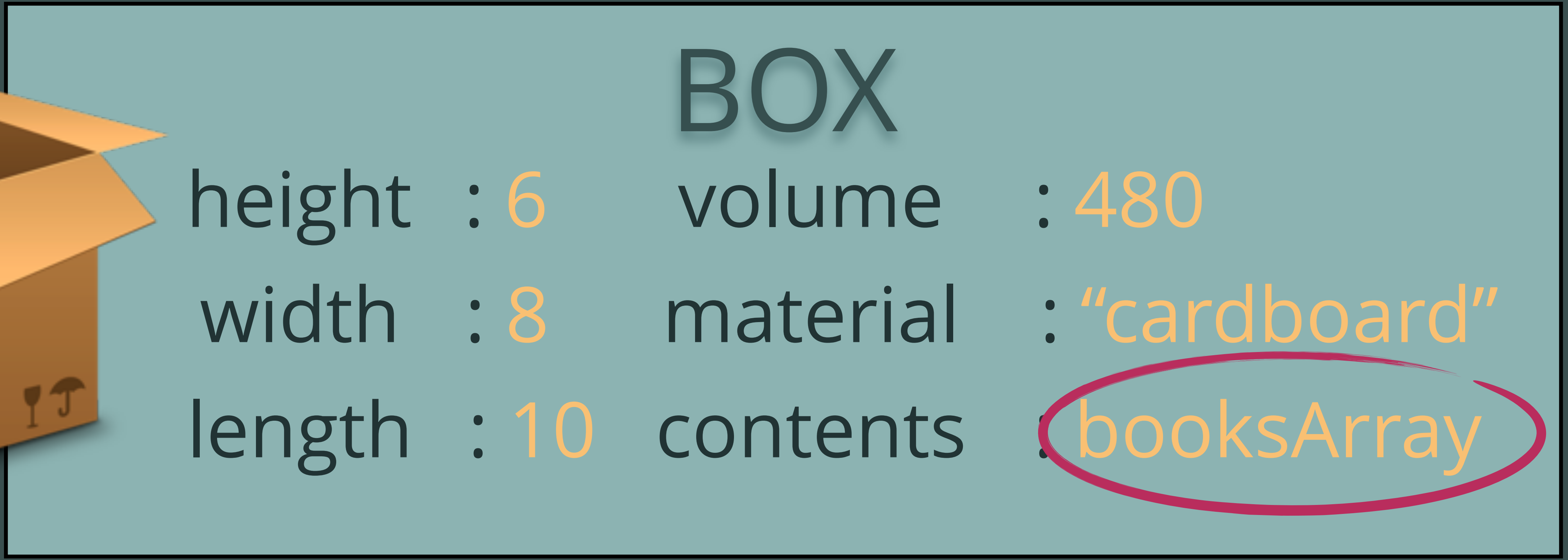
height	: 6	volume	: 480
width	: 8	material	: “cardboard”
length	: 10	contents	: booksArray

```
var myBox = { height: 6, width: 8, length: 10, volume: 480,
              material: "cardboard",
              contents: ["Great Expectations", "The Remains of the Day", "Peter Pan"]
            };
```

Sweet, a box Object complete with properties!

OBJECT PROPERTIES WILL ALSO ACCEPT VARIABLES

We can initialize the 'contents' property with a booksArray variable



BOX

height	: 6	volume	: 480
width	: 8	material	: "cardboard"
length	: 10	contents	: booksArray

```
var myBox = { height: 6, width: 8, length: 10, volume: 480,
              material: "cardboard",
              contents: ["Great Expectations", "The Remains of the Day", "Peter Pan"]
            };
```

OBJECT PROPERTIES WILL ALSO ACCEPT VARIABLES

We can initialize the ‘contents’ property with a booksArray variable

BOX

height	: 6	volume	: 480
width	: 8	material	: “cardboard”
length	: 10	contents	: booksArray

```
var booksArray = ["Great Expectations", "The Remains of the Day", "Peter Pan"];
var myBox = { height: 6, width: 8, length: 10, volume: 480,
              material: "cardboard",
              contents: ["Great Expectations", "The Remains of the Day", "Peter Pan"]
            };
```

OBJECT PROPERTIES WILL ALSO ACCEPT VARIABLES

We can initialize the ‘contents’ property with a booksArray variable

BOX

height	: 6	volume	: 480
width	: 8	material	: “cardboard”
length	: 10	contents	: booksArray

```
var booksArray = ["Great Expectations", "The Remains of the Day", "Peter Pan"];
var myBox = { height: 6, width: 8, length: 10, volume: 480,
              material: "cardboard",
              contents: booksArray
            };
```

REFERENCING AN OBJECT'S PROPERTIES

We can take a peek at any particular property of an object using the dot operator

```
var booksArray = ["Great Expectations", "The Remains of the Day", "Peter Pan"];
var myBox = { height: 6, width: 8, length: 10, volume: 480,
              material: "cardboard",
              contents: booksArray
            };
```

```
myBox.width;
```

```
→ 8
```

```
myBox.materials;
```

```
→ "cardboard"
```

```
myBox.contents;
```

```
→ ["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    8
length:   10
volume:   480
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   480
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   480
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
console.log( myBox.width );
```

→ 12

Oops, that makes our volume incorrect!

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   480
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
console.log( myBox.width );
```

→ 12

```
myBox.volume = myBox.length * myBox.width * myBox.height;
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
console.log( myBox.width );
```

→ 12

```
myBox.volume = myBox.length * myBox.width * myBox.height;
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.width = 12;
console.log( myBox.width );
```

→ 12

```
myBox.volume = myBox.length * myBox.width * myBox.height;
console.log( myBox.volume );
```

→ 720

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

myBox.contents returns an entire Array, to which we can easily apply Array methods.

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan"]
```

myBox

```
height:    6
width:     12
length:    10
volume:    720
material:    "cardboard"
contents:    booksArray
```

```
myBox.contents.push("On The Road");
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

Whoa, we modified the external array outside of myBox ? How'd we do that?

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

Passing in booksArray only makes a REFERENCE to the external Array contained in the variable, and doesn't create a brand new copied Array.

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

Since we're only referring to booksArray, pushing to myBox.contents (or using any Array method) will just modify booksArray!

```
console.log( myBox.contents );
```

```
→ ["Great Expectations", "The Remains of the Day",
   "Peter Pan", "On The Road"]
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

```
console.log( myBox.contents );
```

→

```
["Great Expectations", "The Remains of the Day",
 "Peter Pan", "On The Road"]
```

CHANGING PROPERTY VALUES

The dot operator also allows modification of properties, even using methods

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.contents.push("On The Road");
```

```
console.log( myBox.contents );
```

```
→ ["Great Expectations", "The Remains of the Day",
   "Peter Pan", "On The Road"]
```

```
console.log( booksArray );
```

```
→ ["Great Expectations", "The Remains of the Day",
   "Peter Pan", "On The Road"]
```

ADDING PROPERTY VALUES POST-CREATION

Even after an object has been created, properties can continue to be added

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
```

```
myBox.weight = 24;
```

ADDING PROPERTY VALUES POST-CREATION

Even after an object has been created, properties can continue to be added

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
weight:   24
```

```
myBox.weight = 24;
```

The myBox Object looks around for a weight property. Finding none, it creates one!

ADDING PROPERTY VALUES POST-CREATION

Even after an object has been created, properties can continue to be added

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
weight:   24
destination1:  "Orlando"
destination2:  "Miami"
```

```
myBox.weight = 24;
```

```
myBox.destination1 = "Orlando";
```

```
myBox.destination2 = "Miami";
```

A SECOND WAY OF ACCESSING OR CREATING PROPERTIES

We can use brackets on Objects in similar fashion to accessing array indices

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:    "cardboard"
contents:    booksArray
weight:   24
destination1:  "Orlando"
destination2:  "Miami"
```

```
myBox["volume"];
```

→ 720

```
myBox["material"];
```

→ "cardboard"

An object is like an Array whose indices can be accessed with strings (with quotes) instead of numbers.

A SECOND WAY OF ACCESSING OR CREATING PROPERTIES

We can use brackets on Objects in similar fashion to accessing array indices

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6
width:    12
length:   10
volume:   720
material:   "cardboard"
contents:   booksArray
weight:   24
destination1:  "Orlando"
destination2:  "Miami"
```

```
myBox["# of stops"] = 2;
```

Since the brackets use or "check for" an exactly matching string, we can also create properties with spaces and characters in their names.

A SECOND WAY OF ACCESSING OR CREATING PROPERTIES

We can use brackets on Objects in similar fashion to accessing array indices

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
contents:   booksArray
weight:  24
destination1:  "Orlando"
destination2:  "Miami"
"# of stops":  2
```

```
myBox["# of stops"] = 2;
```

Since the brackets use or "check for" an exactly matching string, we can also create properties with spaces and characters in their names.

A SECOND WAY OF ACCESSING OR CREATING PROPERTIES

We can use brackets on Objects in similar fashion to accessing array indices

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
contents:   booksArray
weight:  24
destination1:  "Orlando"
destination2:  "Miami"
"# of stops":  2
```

```
myBox["# of stops"] = 2;
```

```
console.log( myBox."# of stops" );
```

✖

→ ERROR

No such syntax. Can't put a string after a dot. Beware!

A SECOND WAY OF ACCESSING OR CREATING PROPERTIES

We can use brackets on Objects in similar fashion to accessing array indices

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6   width:    12
length:   10  volume:   720
material:   "cardboard"
contents:   booksArray
weight:  24
destination1:  "Orlando"
destination2:  "Miami"
"# of stops":  2
```

```
myBox["# of stops"] = 2;
```

```
console.log( myBox."# of stops" );
```

✗

→ ERROR

```
console.log( myBox["# of stops"] );
```

✓

→ 2

Thus, key names with spaces can only be accessed with brackets!

BRACKETS ENABLE DYNAMIC PROPERTY ACCESS

Since brackets take expressions, we can avoid hard-coding every property access

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
contents:   booksArray
weight:  24
destination1:  "Orlando"
destination2:  "Miami"
"# of stops":  2
```

```
for(var i = 1, i <= myBox["# of stops"]; i++){
    console.log( myBox["destination" + i] );
}
```

→ Orlando

→ Miami

We can place string-based expressions in the brackets to construct specific property names.

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

Each book in our ‘contents’ property could be a Book object

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
contents:   booksArray
weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

First, we’ll delete our contents property with the delete keyword.

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6   width:    12
length:   10  volume:  720
material:   "cardboard"
~~contents:~~   booksArray
weight:  24
destination1:  "Orlando"
destination2:  "Miami"
"# of stops":  2
```

```
delete myBox.contents;
```

The delete keyword will completely delete the entire contents property...not just the value associated with that property.

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

First, we'll delete our contents property with the delete keyword.

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"

weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

```
delete myBox.contents;
```

→ true

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

First, we'll delete our contents property with the delete keyword.

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:   6   width:    12
length:  10   volume:  720
material:   "cardboard"
weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

```
delete myBox.contents;
```

→ true

```
console.log( booksArray );
```

```
→ ["Great Expectations", "The Remains of the Day",
   "Peter Pan", "On The Road"]
```

Additionally, we've only deleted the property name and the reference, but not the original booksArray *outside the Box.*

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

First, we'll delete our contents property with the delete keyword.

booksArray

```
["Great Expectations", "The Remains of the Day", "Peter Pan", "On The Road"]
```

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

```
delete myBox.contents;
```

→ true

```
delete myBox.nonexistentProperty;
```

→ true

Watch out, though...delete will return true each time, regardless of whether the property existed or not! Think of it as asking: is this property gone?

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

Now, we'll build a function that creates Book objects and adds them to our Box

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

Now, we'll build a function that creates Book objects and adds them to our Box

myBox

```
height:  6   width:   12
length:  10  volume:  720
material:   "cardboard"
weight:  24
destination1: "Orlando"
destination2: "Miami"
"# of stops": 2
```

CHANGING OUR CONTENTS TO INDIVIDUAL OBJECTS

Now, we'll build a function that creates Book objects and adds them to our Box

myBox

```
height: 6   length: 10   width: 12   volume:  720   weight:  24
material: "cardboard"   destination1: "Orlando"   destination2: "Miami"
"# of stops": 2
```