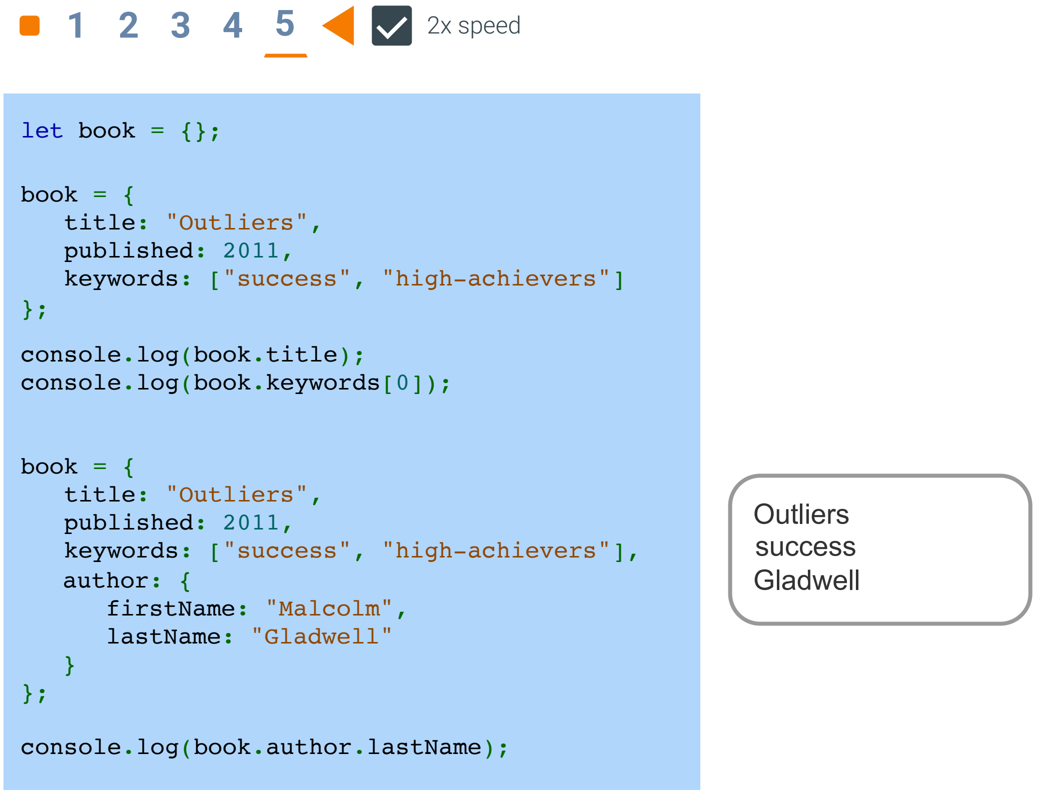
6.9 Objects

Objects and properties

An ***object*** is an unordered collection of properties. An object ***property*** is a name-value pair, where the name is an identifier and the value is any data type. Objects are often defined with an object literal. An ***object literal*** (also called an ***object initializer***) is a comma-separated list of property name and value pairs.

PARTICIPATION ACTIVITY 6.9.1: Creating an object with an object literal.



```
let book = {};

book = {
   title: "Outliers",
   published: 2011,
   keywords: ["success", "high-achievers"]
};

console.log(book.title);
console.log(book.keywords[0]);


book = {
   title: "Outliers",
   published: 2011,
   keywords: ["success", "high-achievers"],
   author: {
      firstName: "Malcolm",
      lastName: "Gladwell"
   }
};

console.log(book.author.lastName);
```

```
Outliers
success
Gladwell
```

Display the last name of the book's author.

Captions

1. book is assigned an empty object literal.
2. book is assigned an object literal with three properties: title, published, keywords.
3. Display the title and first keyword of the book object.
4. book is assigned an object literal with an embedded object literal that is assigned to the author property.
5. Display the last name of the book's author.

Feedback?

PARTICIPATION ACTIVITY | 6.9.2: Accessing object properties. ✓

Refer to the `book` object below.

```
let book = {
   title: "Hatching Twitter",
   published: 2013,
   keywords: ["origins", "betrayal", "social media"],
   author: {
      firstName: "Nick",
      lastName: "Bilton"
   }
};
```

1) Which statement changes the published year to 2014?

- ○ `book.Published = 2014;`
- ● `book.published = 2014;`
- ○ `book.published : 2014;`

Correct

Object property values may be modified after declaring an object.

✓

2) Which statement adds a new property called "isbn" with the value "1591846013"?

- ● `book.isbn = "1591846013";`
- ○ `isbn = "1591846013";`
- ○ `book.isbn("1591846013");`

Correct

New properties may be added to an object after the object is defined with an object literal.

✓

3) What statement replaces "Nick" with "Jack"?

- ○ `book.author = "Jack";`
- ● `book.author.firstName = "Jack";`
- ○ `book.author.lastName = "Jack";`

Correct

`author` is a property of the `book` object. Assigning the `author` object's `firstName` property with "Jack" replaces the author's previous first name.

✓

4) What is missing from the code below to remove "social media" from the book's keywords? The array method `pop()` removes the last element from an array.

```
_______.pop();
```

○ `keywords`
○ `book.keywords[2]`
● `book.keywords`

Correct

`book.keywords.pop()` removes the last item from the `keywords` array.

Feedback?

Methods

Assigning an object's property name with an anonymous function creates a method. Methods access the object's properties using the keyword `this`, followed by a period, before the property name. Ex: `this.someProperty`.

Figure 6.9.1: Defining a method in an object literal.

```
let book = {
   title: "Quiet",
   author: {
      firstName: "Susan",
      lastName: "Cain"
   },

   // Define a method
   getAuthorName: function() {
      return this.author.firstName + " " +
this.author.lastName;
   }
};

// Call a method that returns "Susan Cain"
let name = book.getAuthorName();
```

Feedback?

Figure 6.9.2: Defining a method for an existing object.

```
let book = {
   title: "Quiet",
   author: {
      firstName: "Susan",
      lastName: "Cain"
   }
};

// Define a method
book.getAuthorName = function() {
   return this.author.firstName + " " +
this.author.lastName;
};

// Call a method that returns "Susan Cain"
let name = book.getAuthorName();
```

Feedback?

PARTICIPATION ACTIVITY 6.9.3: Object methods. ✓

Refer to the above figures.

1) A method may be defined inside or outside an object literal.

[✓ True] False

Correct

An object property or method can be assigned after the object is created. Ex: The `getAuthorName()` method in the figure above is defined inside and outside the `book` object.

2) The method below outputs "I'm reading 'Quiet'.".

```
book.read =
function() {

console.log("I'm
reading '" +
title + "'.");
};
```

True [✓ False]

Correct

The `title` variable is not defined. `this.title` is the proper way to access the `title` property in a method.

3) The method below creates a new object property.

```
book.assignMiddleInitial =
function(middleInitial) {
   this.author.middleInitial =
middleInitial;
};

book.assignMiddleInitial("H");
```

[✓ True] False

Correct

The method creates a new property called `author.middleInitial`.

Feedback?

PARTICIPATION ACTIVITY 6.9.4: Practice creating objects and methods.

Create an object called `game` that represents a competition between two opponents or teams. Add the following properties to `game`, and assign any value to each property:

1. `winner` - An object with properties `name` and `score`
2. `loser` - An object with properties `name` and `score`

Add the following methods to `game`:

1. `getMarginOfVictory()` - Returns the difference between the winner's score and the loser's score
2. `showSummary()` - Outputs to the console the winner's name and score, the loser's name and score, and the margin of victory

Call the two methods to verify the methods work correctly. Example output:

```
Broncos: 24
Panthers: 10
Margin of victory: 14
```

JavaScript CSS

```
// Declare a game object and call the game object's methods

```

Run JavaScript

Reset code

Your console output

▶ View solution

Feedback?

Accessor properties

An object property may need to be computed when retrieved, or setting a property may require executing some code to perform data validation. The `get` and `set` keywords define getters and setters for a property.

- A ***getter*** is a function that is called when an object's property is retrieved. Syntax to define a getter: `get property() { return someValue; }`.
- A ***setter*** is a function that is called when an object's property is set to a value. Syntax to define a setter: `set property(value) { ... }`.

An ***accessor property*** is an object property that has a getter or a setter or both.

Figure 6.9.3: Defining an accessor property called 'area'.

```
let rectangle = {
   width: 5,
   height: 8,
   get area() {
      return this.width * this.height;
   },
   set area(value) {
      // Set width and height to the square root of the value
      this.width = Math.sqrt(value);
      this.height = this.width;
   }
};

let area = rectangle.area;      // Calling getter returns 40
rectangle.area = 100;           // Calling setter sets width and
height to 10
console.log(rectangle.width);   // 10
```

Feedback?

PARTICIPATION ACTIVITY | 6.9.5: Accessor properties.

Refer to the `game` object.

```
let game = {
   firstOpponent: "Serena Williams",
   firstOpponentScore: 2,
   secondOpponent: "Garbine Muguruza",
   secondOpponentScore: 0,
   get winner() {
      if (this.firstOpponentScore > this.secondOpponentScore) {
         return this.firstOpponent;
      }
      else if (this.secondOpponentScore > this.firstOpponentScore) {
         return this.secondOpponent;
      }
      else {
         return "Tie";
      }
   }
};
```

1) The code below outputs "Serena Williams".

```
console.log(game.winner());
```

True [✓ False]

✓ **Correct**

`game.winner` is not a method. The proper syntax is: `console.log(game.winner);`

2) The code below outputs "Maria Sharapova" .

```
game.winner = "Maria
Sharapova";
console.log(game.winner);
```

True [✓ False]

✓ **Correct**

`set winner(value) { ... }` is not defined, so `game.winner` may not be set to any other value.

3) The `matchDate` setter below sets the `date` property to the given `value`.

```
let game = {
   ...
   date: "",
   set
matchDate(value)
{
      date =
value;
   },
   ...
};
```

True [✓ False]

✓ **Correct**

The keyword `this` is missing, so a global variable `date` is set, not the `date` property. The correct syntax in the setter is: `this.date = value;`

4) What sets the game's match date to the Date object?

```
let game = {
   ...
   date: "",
   set matchDate(value) {
      this.date = value;
   },
   ...
};

// Wimbledon 2016 women's
championship
let champDate = new
Date(2016, 5, 9);
```

[✓ game.matchDate = champDate;]

game.matchDate(champDate);

Correct

The `matchDate` setter is called when a value is assigned to `matchDate`.

Feedback?

PARTICIPATION ACTIVITY 6.9.6: Practice creating accessor properties.

The `musicQueue` object contains a `songs` property listing all the songs in the music queue. Add an accessor property called "next" with the following functions::

- Getter - Returns the song in the `songs` array at index `nextSong`. Then increments `nextSong` by one so the next song in the queue will be retrieved the next time the getter is accessed. If `nextSong` is beyond the boundaries of the `songs` array, `nextSong` should be assigned 0.
- Setter - Sets `nextSong` to the given value. If the value is outside the `songs` array's bounds, `nextSong` should be assigned 0.

If the `next` property is implemented correctly, the for loop under the `musicQueue` will display each song three times. The code under the for loop tests the setter and should display the song in comments.

```
   // Add getter and setter for next property
};

// Run through the queue three times
for (let c = 0; c < musicQueue.songs.length * 3; c++) {
   console.log("Now playing: " + musicQueue.next);
}

// Test the next setter
musicQueue.next = 2;
console.log(musicQueue.next);    // Macarena
musicQueue.next = 3;
console.log(musicQueue.next);    // Party Rock Anthem
musicQueue.next = -1;
console.log(musicQueue.next);    // Party Rock Anthem

```

Run JavaScript   Reset code

Your console output

```
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
Now playing: undefined
2
3
-1
```

▶ View solution

Feedback?

Passing objects to functions

JavaScript data types can be divided into two categories: primitives and references.

1. A ***primitive*** is data that is not an object and includes no methods. Primitive types include: boolean, number, string, null, and undefined.
2. A ***reference*** is a logical memory address. Only objects are reference types.

Assigning a variable with a primitive creates a copy of the primitive. Ex: If y is 2, then `x = y;` means x is assigned with a copy of y. Assigning a variable with a reference creates a copy of the reference. Ex: If y refers to an object, then `x = y;` means x is assigned with a copy of y's reference. Both x and y refer to the same object.

When a primitive is passed to a function, the parameter is assigned a copy of the argument. Changing the parameter does not change the argument.

When an object is passed to a function, the parameter is also assigned a copy of the argument. However, the parameter and argument are both a reference to the same object. Changing the parameter reference to a new object does not change the argument, but changing the parameter's properties *does* change the argument's properties.

Primitive wrappers

All primitives, except for null and undefined, have equivalent objects that wrap the primitive values and provide methods for interacting with the primitives. Ex: A string primitive has a `String` class that provides various methods for manipulating a string. Calling `"abc".toUpperCase()` converts the primitive string into a `String` object, calls the method, and returns the string primitive "ABC".

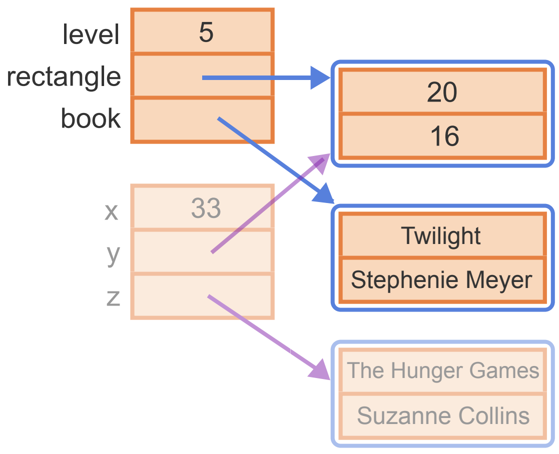
PARTICIPATION ACTIVITY 6.9.7: Passing primitives and references to a function.

1 2 3 4 5 6 7 2x speed

```
function changeThings(x, y, z) {
   x = 33;
   y.width = 20;
   z = { title: "The Hunger Games",
         author: "Suzanne Collins" };
}

let level = 5;
let rectangle = {
   width: 3,
   height: 16
};
let book = {
   title: "Twilight",
   author: "Stephenie Meyer"
};

changeThings(level, rectangle, book);
console.log(level);
console.log(rectangle.width);
console.log(book.title);
```



```
5
20
Twilight
```

After returning from changeThings(), rectangle.width is the only value that has change

Captions

1. level is a number, which is a primitive type.
2. rectangle and book are objects. Each object refers to the object's location in memory.
3. The call to changeThings() assigns a copy of each argument to the x, y, and z parameters. y refers to the same object as rectangle, and z refers to the same object as book.
4. Assigning x a new number does not change level.
5. Assigning y.width a new number changes rectangle.width since both y and rectangle refer to the same object.
6. Assigning z a new object does not change book since z and book refer to different objects.
7. After returning from changeThings(), rectangle.width is the only value that has changed.

Feedback?

PARTICIPATION ACTIVITY 6.9.8: Passing objects to functions. ✓

Refer to the code below.

```
function changeMovie(movie) {
   movie.title = "The Avengers";
   movie.released = 2012;
   movie = {
      title: "Avengers: Endgame",
      released: 2019 };
}

let avengersMovie = {
   title: "Avengers: Infinity War",
   released: 2018
};
```

1) What is output to the console?

```
changeMovie(avengersMovie);
console.log(avengersMovie.title);
```

- ○ Avengers: Infinity War
- [✓ The Avengers]
- ○ Avengers: Endgame

Correct ✓

`changeMovie()` assigns `movie.title` "The Avengers", which changes `avengersMovie.title`. When `movie` is assigned a new movie, `avengersMovie` is not affected.

2) What is output to the console?

```
let myMovie = avengersMovie;
myMovie.title = "Avengers: Age of
Ultron";
console.log(avengersMovie.title);
```

Correct ✓

Assigning `myMovie` the object `avengersMovie` makes `myMovie` and `avengersMovie` refer to the same object in memory. Changing a

○ Avengers: Infinity War
○ The Avengers
● Avengers: Age of Ultron

myMovie property changes the same **avengersMovie** property .

Feedback?

CHALLENGE ACTIVITY | 6.9.1: Objects.

530096.4000608.qx3zqy7

Jump to level 1

✓ 1
✓ 2
3
4
5

Define a method named orderOfAppearance() that takes the name of a role as an argument and returns that role's order of appearance. If the role is not found, the method returns 0. Ex: orderOfAppearance("Elizabeth Swann") returns 3. Hint: A method may access the object's properties using the keyword this. Ex: this.title accesses the object's title property.

```
   director: "Gore Verbinski",
   composer: "Hans Zimmer",
   roles: [ // Roles are stored in order of appearance
      "Jack Sparrow",
      "Will Turner",
      "Elizabeth Swann",
      "Hector Barbossa"
   ],
   orderOfAppearance: function(role) {

      /* Your solution goes here */
      let roleIndex = this.roles.indexOf(role);

      // If the role is found, return its order of appearance
      // If not found, return 0
      return roleIndex !== -1 ? roleIndex + 1 : 0;
```

1 **2** 3 4 5

Check Next

✓

✓ Testing orderOfAppearance("Elizabeth Swann")

Yours 3

✓ Testing orderOfAppearance("Saruman")

Yours 0

Feedback?

Exploring further:

- Working with objects (MDN)

How was this section?

Provide section feedback