COURSE TECHNOLOGY
CENGAGE Learning™

JavaScript, Sixth Edition

Chapter 7

Using Object-Oriented JavaScript

Objectives

When you complete this chapter, you will be able to:

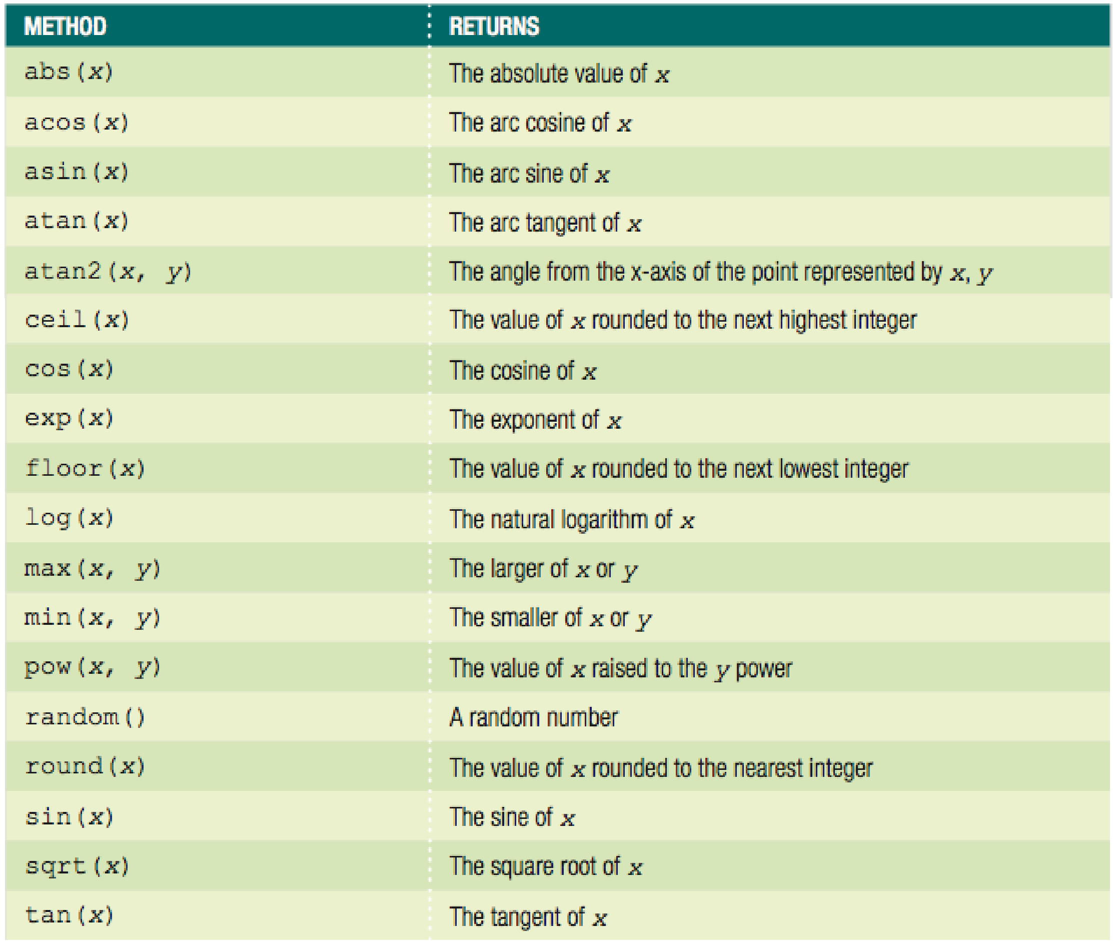
- Explain basic concepts related to object-oriented programming
- Use the `Date`, `Number`, and `Math` objects
- Define your own custom JavaScript objects

Performing Math Functions with the `Math` Class

- `Math` class
  - Methods and properties for mathematical calculations
- Cannot instantiate a Math object using a statement such as: `var mathCalc = new Math();`
  - Use the `Math` object and one of its methods or properties directly in the code
- Example:

```
var curNumber = 144;
var squareRoot = Math.sqrt(curNumber); // returns 12
```

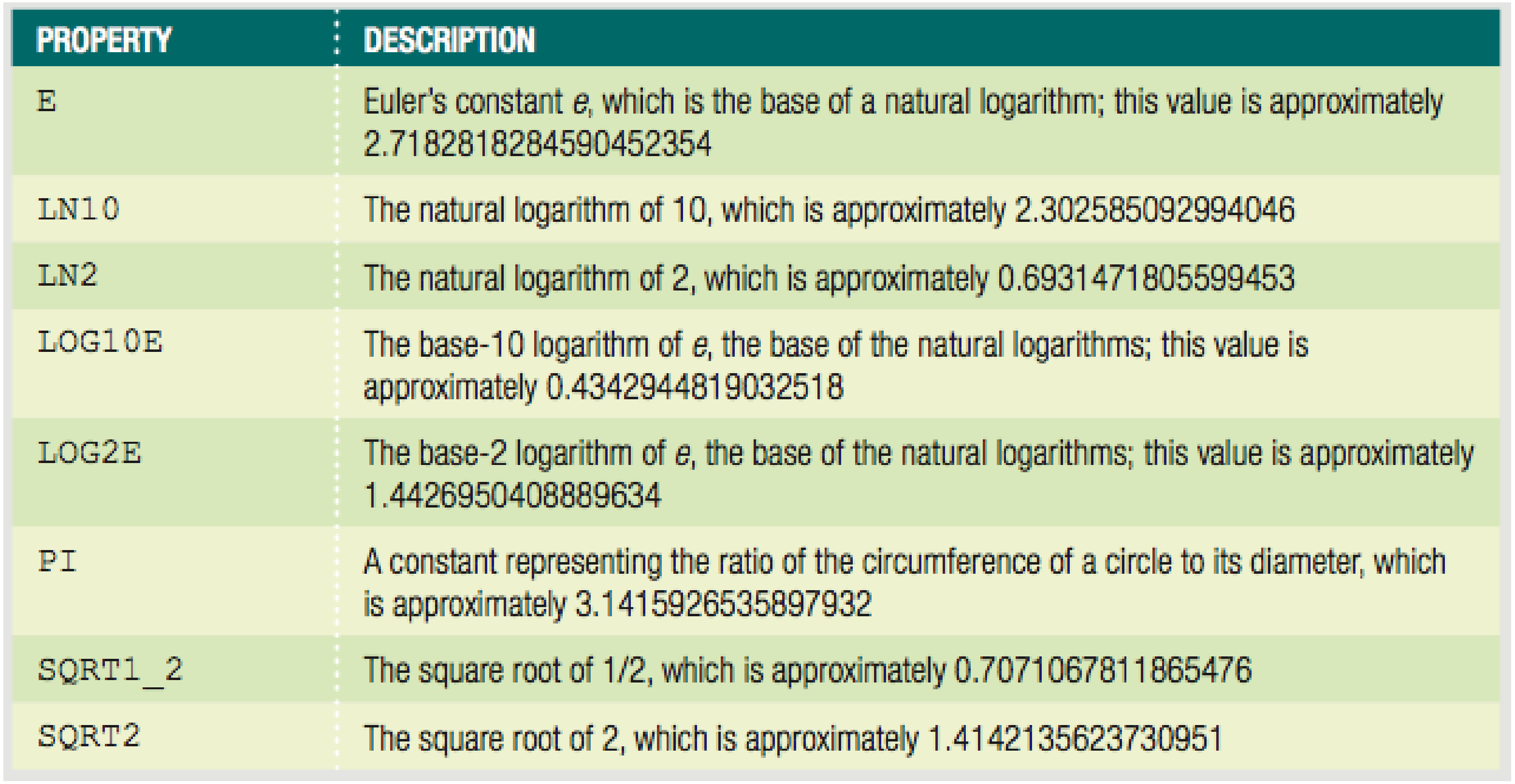
Performing Math Functions with the `Math` Class (cont'd.)



METHOD	RETURNS
`abs(x)`	The absolute value of *x*
`acos(x)`	The arc cosine of *x*
`asin(x)`	The arc sine of *x*
`atan(x)`	The arc tangent of *x*
`atan2(x, y)`	The angle from the x-axis of the point represented by *x*, *y*
`ceil(x)`	The value of *x* rounded to the next highest integer
`cos(x)`	The cosine of *x*
`exp(x)`	The exponent of *x*
`floor(x)`	The value of *x* rounded to the next lowest integer
`log(x)`	The natural logarithm of *x*
`max(x, y)`	The larger of *x* or *y*
`min(x, y)`	The smaller of *x* or *y*
`pow(x, y)`	The value of *x* raised to the *y* power
`random()`	A random number
`round(x)`	The value of *x* rounded to the nearest integer
`sin(x)`	The sine of *x*
`sqrt(x)`	The square root of *x*
`tan(x)`	The tangent of *x*

Table 7-6 Math class methods

Performing Math Functions with the `Math` Class (cont’d.)



PROPERTY	DESCRIPTION
`E`	Euler's constant *e*, which is the base of a natural logarithm; this value is approximately 2.7182818284590452354
`LN10`	The natural logarithm of 10, which is approximately 2.302585092994046
`LN2`	The natural logarithm of 2, which is approximately 0.6931471805599453
`LOG10E`	The base-10 logarithm of *e*, the base of the natural logarithms; this value is approximately 0.4342944819032518
`LOG2E`	The base-2 logarithm of *e*, the base of the natural logarithms; this value is approximately 1.4426950408889634
`PI`	A constant representing the ratio of the circumference of a circle to its diameter, which is approximately 3.1415926535897932
`SQRT1_2`	The square root of 1/2, which is approximately 0.7071067811865476
`SQRT2`	The square root of 2, which is approximately 1.4142135623730951

Table 7-7 Math class properties

Performing Math Functions with the `Math` Class (cont'd.)

- Example:
  - Use the `PI` property to calculate the area of a circle based on its radius
    - Code uses the `pow()` method to raise the radius value to second power, and the `round()` method to round the value returned to the nearest whole number

```
var radius = 25;
var area = Math.PI * Math.pow(radius, 2);
var roundedArea = Math.round(area); // returns 1963
```

Defining Custom JavaScript Objects

- JavaScript: not a true object-oriented programming language
  - Cannot create classes in JavaScript
  - Instead, called an object-based language
- Can define custom objects
  - Not encapsulated
  - Useful to replicate the same functionality an unknown number of times in a script

Declaring Basic Custom Objects

- Use the `Object` object
  - var *objectName* = new Object();
  - var *objectName* = {};
- Can assign properties to the object
  - Append property name to the object name with a period

Declaring Basic Custom Objects (cont'd.)

- Add properties using dot syntax
  - Object name followed by dot followed by property name
  - Example:

```
InventoryList.inventoryDate = new Date(2017, 11, 31);
```

Declaring Basic Custom Objects (cont'd.)

- Can assign values to the properties of an object when object first instantiated
- Example:

```
var PerformanceTickets = {
   customerName: "Claudia Salomon",
   performanceName: "Swan Lake",
   ticketQuantity: 2,
   performanceDate: new Date(2017, 6, 18, 20)
};
```

Declaring Sub-Objects

- Value of a property can be another object
  - called a sub-object
  - Example–`order` object with `address` sub-object:

```
var order = {
   orderNumber: "F5987",
   address: {
      street: "1 Main St",
      city: "Farmington",
      state: "NY",
      zip: "14425"
   }
};
```

Referring to Object Properties as Associative Arrays

- Associative array
  - An array whose elements are referred to with an alphanumeric key instead of an index number
- Can also use associative array syntax to refer to the properties of an object
- With associative arrays
  - Can dynamically build property names at runtime

Referring to Object Properties as Associative Arrays (cont'd.)

- Can use associative array syntax to refer to the properties of an object
- Example:

```
var stopLightColors = {
   stop: "red",
   caution: "yellow",
   go: "green"
};
stopLightColors["caution"];
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers
  - Example:

```
var order = {
   item1: "KJ2435J",
   price1: 23.95,
   item2: "AW23454",
   price2: 44.99,
   item3: "2346J3B",
   price3: 9.95
};
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can easily reference property names that contain numbers (cont'd.)
  - To create order summary:

```
for (var i = 1; i < 4; i++) {
   document.getElementById("itemList").innerHTML +=↵
      "<p class='item'>" + order["item" + i] + "</p>";
   document.getElementById("itemList").innerHTML +=↵
      "<p class='price'>" + order["price" + i] + "</p>";

};
```

Referring to Object Properties as Associative Arrays (cont'd.)

- Can also write generic code to add new object properties that incorporate numbers
  - Example—adding items to shopping cart:

```
totalItems += 1; // increment counter of items in order
currentItem = document.getElementById("itemName").innerHTML;
currentPrice = document.getElementById("itemPrice").innerHTML;
newItemPropertyName = "item" + totalItems; // "item4"
newPricePropertyName = "price" + totalItems; // "price4"
order.newItemPropertyName = currentItem; // order.item4 = (name)
order.newPricePropertyName = currentPrice;
// order.price4 = (price);
```

- Allows for as many items as user wants to purchase

Creating Methods

- Object method simply a function with a name within the object
- Two ways to add method to object
  - Provide code for method in object
  - Reference external function

Creating Methods (cont'd.)

- Specify method name with anonymous function as value
  - Example:

```
var order = {
   items: {},
   generateInvoice: function() {
      // function statements
   }
};
```

Creating Methods (cont'd.)

- Specify method name with existing function as value
  - Example:

```
function processOrder() {
   // function statements
}
var order = {
   items: {},
   generateInvoice: processOrder
};
```

  - Reference to existing function cannot have parentheses

Enumerating custom object properties

- Custom objects can contain dozens of properties
- To execute the same statement or command block for all the properties within a custom object
  - Use the `for/in` statement
  - Looping statement similar to the `for` statement
- Syntax

```
for (variable in object) {
   statement(s);
}
```

Enumerating custom object properties (cont'd.)

- `for/in` statement enumerates, or assigns an index to, each property in an object
- Typical use:
  - validate properties within an object

Enumerating custom object properties (cont’d.)

- Example—checking for empty values:

```
var item={
   itemNumber: "KJ2435J",
   itemPrice: 23.95,
   itemInstock: true,
   itemShipDate: new Date(2017, 6, 18),
};
for (prop in order) {
   if (order[prop] === "") {
      order.generateErrorMessage();
   }
}
```

Deleting Properties

- Use the `delete` operator
- Syntax

  `delete` *`object.property`*
- Example:

```
delete order.itemInStock;
```

Defining Constructor Functions

- Constructor function
  - Used as the basis for a custom object
  - Also known as object definition
- JavaScript objects
  - Inherit all the variables and statements of the constructor function on which they are based
- All JavaScript functions
  - Can serve as a constructor

Defining Constructor Functions (cont’d.)

- Example:
  - Define a function that can serve as a constructor function

```
function Order(number, order, payment, ship) {
   this.customerNumber = number;
   this.orderDate = order;
   this.paymentMethod = payment;
   this.shippingDate = ship;
}
```

Adding Methods to a Constructor Function

- Can create a function to use as an object method
  - Refer to object properties with `this` reference
  - Example:

```
function displayOrderInfo() {
   var summaryDiv = document.getElementById("summarySection");
   summaryDiv.innerHTML += ("<p>Customer: " +↵
      this.customerNumber + "</p>");
   summaryDiv.innerHTML += ("<p>Order Date: " +↵
      this.orderDate.toLocaleString()+ "</p>");
   summaryDiv.innerHTML += ("<p>Payment: " +↵
      this.paymentMethod + "</p>");
   summaryDiv.innerHTML += ("<p>Ship Date: " +↵
      this.shippingDate.toLocaleString() + "</p>");
}
```

Using the `prototype` Property

- After instantiating a new object
  - Can assign additional object properties
    - Use a period
- New property only available to that specific object
- `prototype` property
  - Built-in property that specifies the constructor from which an object was instantiated
  - When used with the name of the constructor function
    - Any new properties you create will also be available to the constructor function

Using the `prototype` Property (cont'd.)

- Object definitions can use the `prototype` property to extend other object definitions
  - Can create a new object based on an existing object

Summary

- Object-oriented programming (or OOP)
  - The creation of reusable software objects
- Reusable software objects
  - Called components
- Object
  - Programming code and data treated as an individual unit or component
- Objects are encapsulated
- Interface represents elements required for a source program to communicate with an object

Summary (cont’d.)

- Principle of information hiding
- Code, methods, attributes, and other information that make up an object
  - Organized using classes
- Instance
  - Object created from an existing class
- An object inherits the characteristics of the class on which it is based
- Date class contains methods and properties for manipulating the date and time

Summary (cont’d.)

- `Number` class contains methods for manipulating numbers and properties
- `Math` class contains methods and properties for performing mathematical calculations
- Can define custom object
  - object literal
- Can create template for custom objects
  - constructor function
- `this` keyword refers to object that called function
- `prototype` property specifies object's constructor