



INTRO TO OBJECTS

WHAT ARE OBJECTS?

- We've been mentioning them all semester
- They are **containers** that hold multiple pieces of information
 - Properties
 - Functionalities called Methods
- We can access the properties and methods of an object using the **dot operator**

`document.getElementById()`



Object



Method

`event.keyCode`



Object



Property

JAVASCRIPT AND OBJECTS

- JavaScript is an object oriented language
- Many of the built in functions we have been using are provided to us via an object oriented interface
- The web browser interacts with JavaScript using an object oriented interface called the DOM (Document Object Model)
- We have been using JavaScript to modify our HTML pages
 - i.e., displaying total costs or lists of info directly on the webpage

THE DOCUMENT OBJECT

- Often when we have had our JavaScript interact with the HTML we have used `document.getElementById`
- We can use this because of the document object
- The document object represents the entire webpage, so we obtain references to individual elements on the page using `getElementById()` method

```
let element = document.getElementById("div1") ;
```

ELEMENT OBJECTS

- Element objects are accessed through their id
- Each object has many properties and methods that we can use
- Ex. innerHTML property

```
let element = document.getElementById("div1") ;
```

- Ex. If we want to replace all HTML inside the tags:

```
element.innerHTML = "<p>Here is some new text</p>"
```

- Ex. If we want to add to the HTML already inside the tags:

```
element.innerHTML += "<p>Some more text</p>"
```



USEFUL OBJECTS

- Some others we have already seen
 - Math Object - allows us to use certain constants and functions
 - Document object - the representation of the entire webpage except the <head>. It contains all other elements on the page
- Some we haven't seen
 - Window object - the current open window of the browser
 - Navigator object - contains information about the browser itself

EXPLORE OBJECTS!

- There are many objects as part of JavaScript with respect the web browser
- Explore the documentation to find the features you need
- You will need to explore on your own a little for your project, as the interface is too rich to be presented in it's entirety in a class
- And not every feature will be relevant to all applications
- <https://developer.mozilla.org/en-US/>
- <http://www.w3schools.com/>

WHY USE OBJECTS?

- Objects allow us to Group data/variables and functionality/functions into one logical package with meaningful names
- Object Properties
 - Are pieces of data that can be:
 - Strings, Numbers, Boolean
 - Other Objects
- Object Methods:
 - Are abilities or functionalities that the object can DO
 - These are special functions that have access to the object's data

MAKING YOUR OWN OBJECTS

- The first step is to **define** your object (create it)
- One way we can create objects is using **JavaScript Object Notation(JSON)**
- We simply make a list of properties and values inside curly braces

```
let myFirstObject =  
{  
  name: "Jamie",  
  height: 180,  
  isEvil: false  
};
```

Properties { Values

MAKING YOUR OWN OBJECTS

- We see that the property name (or variable name) is followed by a colon, then its value
- If there is more than one property we separate them by commas
- Different properties can have different types

```
let myFirstObject =  
{  
  name: "Jamie",  
  height: 180,  
  isEvil: false  
};
```

Properties { Values

EXAMPLE

- Let's consider a problem about weather data. Let's create an object called Tuesday that contains the following weather report information:

Temperature	10 Celsius
Chance of rain	50%

```
let tuesday = {  
  temperature: 10,  
  rainChance: 0.5  
};
```

tuesday.temperature has a value of 10
tuesday.rainChance has a value of 0.5

EXAMPLE

- What if we wanted an object to represent a point?
- How could we create a object representing the point (100,200)?

```
let myPoint = {  
    x: 100,  
    y: 200  
};
```

Then we could access the
x and y locations using:

myPoint.x

and:

myPoint.y

CHANGING PROPERTY VALUES

- To change the value of a property stored in an object we use the assignment operator (as we've seen before)

```
let myPoint = {  
    x: 100,  
    y: 200  
};  
myPoint.x = 200;  
console.log(myPoint.x) ;  
//will now print 200
```

EXERCISE

- Create an object using JavaScript Object Notation
- Give the object an appropriate name and fill it with the following information

Property	Value
Name	Eggs
Price	4.99
Quantity	12

OBJECTS AND REFERENCES

- Objects are assigned by reference

```
let object1 = {  
    name: "object1",  
    date: "today"  
};
```

```
let object2 = object1;
```

- This last assignment cause object1 and object2 to refer to the same object

OBJECTS AND REFERENCES

```
let object1 = {  
    name: "object1",  
    date: "today"  
};  
let object2 = object1;  
object1.name = "otherObject";  
console.log(object1.name);  
console.log(object2.name);
```

what gets printed to the console?

PASS BY REFERENCE

- Unlike primitive types, when objects are passed into functions as parameters they are passed by reference, and not by value

```
function changeDate(someDate) {  
    someDate.date = "tomorrow";  
}  
  
let object1 = {  
    name: "object1",  
    date: "today"  
};  
  
changeDate(object1);  
console.log(object1.date);
```

What gets printed to the console?

OTHER WAY TO CREATE OBJECTS

- Is by using a **constructor**
 - a constructor is a function whose purpose is to set up an object
 - are used when we want to create multiple objects with the same properties and methods
- **Constructors do not have returns! EVER**
- **Notice the capitalization!**

```
function Point(x1, y1) {  
    this.x = x1;  
    this.y = y1;  
}
```

CALLING CONSTRUCTOR

- We use the keyword **new** to create a new generic object, and then the constructor function fills in the values via parameters

```
let p1 = new Point(100,200) ;
```

- If you forget to use the keyword **new** then Point() is just a regular function call, which may have unintended side effects



WHAT'S THIS?

- **this** is a JavaScript keyword that we can use to increase the specificity of our variables and properties
- Outside of a function **this** this refers to the global object, like the window or document object

WHAT'S THIS?

- Whereas when it is a part of a method of an object, this is set to be the object the method is called from/on

```
function Point(x1,y1) {  
    this.x = x1;  
    this.y = y1;  
}  
let firstPoint = new Point(10,50);
```



CHANGE PROPERTY VALUES

```
function Point(x1,y1){  
    this.x = x1;  
    this.y = y1;  
}  
let firstPoint = new Point(10,50);  
firstPoint.x = 200;  
console.log(firstPoint.x);
```

PARAMETERS

- Constructors accept parameters like any regular function does
- This is great for creating objects of the same type, but that have different property values
- Why would we create the same point over and over again like we have been doing?

```
function Point(x1, y1) {
```

```
    this.x = x1;
```

```
    this.y = y1;
```

```
}
```

```
let firstPoint = new Point(100, 200);
```

```
let secondPoint = new Point(200, 300);
```


EXAMPLE

- To create several of the same object with unique properties

```
function Car(make, model, doors, color) {  
    this.make = make;  
    this.model = model;  
    this.doors = doors;  
    this.color = color;  
}  
  
let myCar = new Car("Honda", "Fit", 4, "grey");  
let dreamCar = new Car("Tesla", "Model3", 4, "blue");
```

NOTE:

```
let myCar = new Car("Honda", "Fit", 4, "grey");
```

- So, the new keyword creates a new object
- The statement on the right hand side evaluates to an object, after the constructor function finishes executing
 - This is because of the new keyword
 - If we just ran the constructor function without the new keyword, it would not evaluate to an object
- That object reference is then assigned to the variable myCar

EXERCISE

- Create a constructor that can be used to create an object that represents a store with the properties in the table below.

Property	Description
Name	The name of the store
Staff	The number of employees the store has
Owner	Name of the store owner

- Why does this question not have a value column when the previous exercise did?

ARRAYS OF OBJECTS

- Last week we started building arrays of primitive types
- This week we've started talking about objects
- Let's talk about arrays of objects!
- Remember our point constructor:

```
function Point(x, y) {  
    this.x = x;  
    this.y = y;  
}
```

CREATING AN ARRAY OF POINTS

```
let pointList = [new Point(100,400),  
                 new Point(200,500),  
                 new Point(300,600)];
```

- creates the following list of points

```
pointList => [{x: 100, y: 400},  
              {x: 200, y: 500},  
              {x: 300, y: 600}];
```

- that we can access via

```
pointList[0].x is 100
```

```
pointList[0].y is 400
```

CREATING AN ARRAY OF POINTS

```
let pointList = [new Point(100,400),  
                 new Point(200,500),  
                 new Point(300,600)];
```

- creates the following list of points

```
pointList => [{x: 100, y: 400},  
              {x: 200, y: 500},  
              {x: 300, y: 600}];
```

- What are the following values?
- `pointList[1].x`
- `pointList[2].y`

EXERCISE

- Create an array of objects as described below, you can either use a constructor or literal objects (JSON)

Index	Properties	Values
0	Prep time	15 minute
	Cooking time	10 minutes
	Ingredients	["shrimp", "garlic"]
1	Prep time	20 minutes
	Cooking time	15 minutes
	Ingredients	["honey", "salmon", "pepper"];
2	Prep time	30 minutes
	Cooking time	0 minutes
	Ingredients	["banana", "orange", "apple", "grape"]

METHODS

- Methods are functions that can use an object's internal data (like properties)
- Methods should not be run without an object
 - JavaScript is the only language where methods can exist independently from objects
- To access an object's data in a method we use the keyword **this**
- Adding methods to an object is done the same way we add properties

METHOD

```
function Recipe(prepare, cook, ingredients){  
    this.prepare = prepare;  
    this.cook = cook;  
    this.ingredients = ingredients.slice();  
    this.totalTime = function(){  
        return this.prepare + this.cook;  
    }  
}  
  
let spaghetti = new Recipe(5,15,["pasta","sauce"]);  
console.log(spaghetti.totalTime());
```

What gets printed?

EXAMPLE WITH JSON

```
let myStore = {  
  store: "Superstore",  
  staff: 150,  
  owner: "Jim Pattison",  
  toString: function(){  
    return this.store+" "+this.staff+"  
                                     "+this.owner;  
  };  
};  
console.log(myStore.toString())
```

What gets printed?

EXAMPLE WITH JSON

```
let myStore = {  
    store: "Superstore",  
    staff: 150,  
    owner: "Jim Pattison",  
    toString: function() {  
        return this.store+" "+this.staff+"  
"+this.owner;  
    };  
};  
  
console.log(myStore.toString())
```

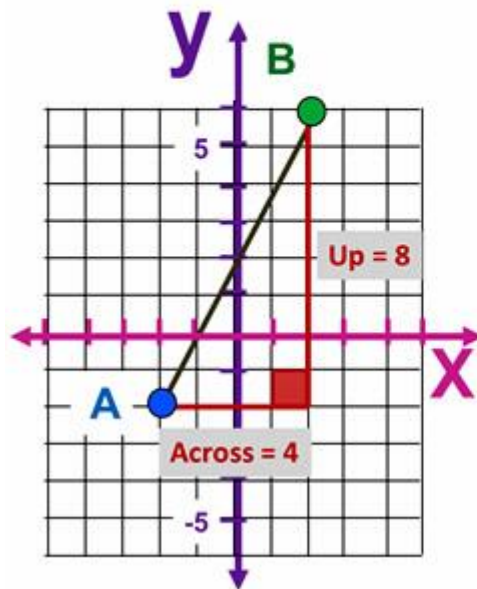
What gets printed?

Prints "Superstore 150 Jim Patterson"

MATH BREAK!

- Remember the distance function?

DISTANCE BETWEEN POINTS



$$(AB)^2 = 4^2 + 8^2$$

$$(AB)^2 = 16 + 64$$

$$(AB)^2 = 80$$

$$AB = \sqrt{80} \text{ or } 8.94 \checkmark$$

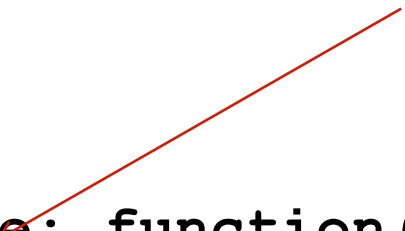
MATH BREAK

- So then to find the distance between two points is calculated as
- $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- So the distance from the origin (0,0) to some point (x,y) can be found using
- $d = \sqrt{(x_2 - 0)^2 + (y_2 - 0)^2}$
- Or
- $d = \sqrt{x^2 + y^2}$
- So let's add a distance method to our point objects!

EXAMPLE

```
let point = {  
  x: 100,  
  y: 100,  
  distance: function() {  
    let xSq = Math.pow(this.x,2);  
    let ySq = Math.pow(this.y,2);  
    return Math.sqrt(xSq + ySq);  
  }  
};
```

How far from (0,0) is (x,y)?



EXAMPLE

```
let point = {  
  x: 100,  
  y: 100,  
  distance: function() {  
    let xSq = Math.pow(this.x,2);  
    let ySq = Math.pow(this.y,2);  
    return Math.sqrt(xSq + ySq);  
  }  
};
```

- To call the method we write something like

```
console.log(point.distance()); Prints 141.421
```

EXAMPLE

- When the point changes, the value returned by the distance function will also change, because the function uses the object's x and y values

```
let point = {  
  x: 200,  
  y: 100,  
  distance: function() {  
    let xSq = Math.pow(this.x,2);  
    let ySq = Math.pow(this.y,2);  
    return Math.sqrt(xSq + ySq);  
  }  
};  
console.log(point.distance());
```

Now prints 223.607

SAME EXAMPLE USING CONSTRUCTOR

```
function Point(x, y) {  
    this.x = x;  
    this.y = y;  
    this.distance = function() {  
        let xSq = Math.pow(this.x, 2);  
        let ySq = Math.pow(this.y, 2);  
        return Math.sqrt(xSq + ySq);  
    }  
}  
  
let myPoint = new Point(200, 200);  
console.log(myPoint.distance());
```

Prints 282.842

EXERCISE

- Modify our distance method to accept two parameters, x2 and y2. This function should then calculate and return the distance between the point stored in the object and the point passed in as a parameter

-

```
function Point(x, y) {  
  this.x = x;  
  this.y = y;  
  this.distance = function() {  
    let xSq = Math.pow(this.x, 2);  
    let ySq = Math.pow(this.y, 2);  
    return Math.sqrt(xSq + ySq);  
  }  
}
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

A QUICK NOTE

- We place semicolons after each statement
- In JavaScript if semicolons are missing, JavaScript will make an educated guess where one should be placed
 - This will sometime lead to subtle bugs if JavaScript guesses wrong
- A program starts executing from the top, and makes it's way to the bottom
- Various things, like function calls, conditionals, loops will change the path of the program

ASSOCIATIVE ARRAYS

- Many programming languages support arrays with named indexes
- Arrays with named indexes are called associative arrays (or hashes)
- JavaScript does **not** support arrays with named indexes
- In JavaScript, **arrays** always use **numbered indexes**
- If you use a named index, JavaScript will redefine the array to a standard object
- After that, **all array methods and properties will produce incorrect results**

RELATIONSHIP: ARRAYS AND OBJECTS

- In JavaScript, **arrays** use **numbered indexes**
- In JavaScript, **objects** use **named indexes**
- Arrays are a special kind of object that has numbered indexes
- JavaScript does not support string indexed arrays
- You should use **objects** when you want the element names to be **strings (text)**
- You should use **arrays** when you want the element names to be **numbers**

EXAMPLE

```
let person = [];
```

```
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;
```

```
var x = person.length;  
//person.length return 0
```

```
var y = person[0];  
//person[0]return undefined
```

HOWEVER - person is now an object!
This means we can use the dot operator



```
person.firstName  
"John"
```

```
person.lastName  
"Doe"
```

```
person.age  
46
```

CLONING OBJECTS

- Remember that example last lecture

```
let object1 = {  
    name: "object1",  
    date: "today"  
};
```

```
let object2 = object1;
```

- This last assignment cause object1 and object2 to refer **to the same object**, so then

```
object1.name = "otherObject";
```

will affect both object1 and object2

- So then how do we copy or clone an object?

OBJECT.ASSIGN()

- The Object.assign() method is used to copy the values of all enumerable own properties from one or more source objects to a target object. It will return the target object.
- It expects two parameters:
- First- The target object, { } for a new object
- Sources - The source object(s) to copy/clone.
- Properties in the target object will be overwritten by properties in the sources if they have the same name

OBJECT.ASSIGN()

```
let object1 = {  
    name: "object1",  
    date: "today"  
};
```

```
let object2 = Object.assign({}, object1);
```

- Now object1 and object2 are two completely separate objects, that just happen to have the same properties and values

EXERCISE

- Create a constructor with the following properties
 - Name - Name of the pet.
 - Species - Species of the pet(eg. cat, dog)
 - Breed - Breed of the pet
 - Allergies - List of allergies for the pet
- Note: For lists, or things that are stored in arrays you need to create a copy of the array parameter
- Now add a method to your constructor
 - isAllergic(item) that returns true if item occurs in the pet's list of allergies

EXERCISE

- Create a constructor for a bank account object
- This object should have a property for the account balance
- The constructor should accept one parameter, and use it to initialize the account balance property
- The constructor should have the following methods
 - getBalance - returns the balance of the account
 - setBalance - accepts an amount as a parameter and sets the balance of the account to be this amount
 - Withdrawal - accepts an amount as a parameter and returns the account balance minus the withdrawal amount
 - Deposit - accepts an amount as a parameter and returns the account balance plus the withdrawal amount