upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: JavaScript and Server-Side Communication

Lecture On: JavaScript Arrays, Objects and Functions

Instructor: Siddhesh Prabhugaonkar

In the previous class, we discussed…

- Introduction to JavaScript (JS)
- How to add JS to HTML files
- Syntax of JS
- Data types in JS
- Operators in JS
- Looping and conditional statements in JS

Poll 1 (15 Sec)

Which of the following is true regarding the ***typeof*** operator in JS?
(Note: More than one option may be correct.)

1. It is used to get the datatype of the operator.
2. It is used to get the datatype of the operand.
3. Its value is a string that represents the data type of the operand.
4. All of the above

Poll 1 (Answer)

Which of the following is true regarding the ***typeof*** operator in JS?
(Note: More than one option may be correct.)

1. It is used to get the datatype of the operator.
2. **It is used to get the datatype of the operand.**
3. **Its value is a string that represents the data type of the operand.**
4. All of the above

Poll 2 (15 Sec)

In which of the following scenarios would ‘switch...case’ be unsuitable?

1. If the signal is green, then turn on a machine, and if the signal is red, then turn off the machine.
2. If a number is equal to 10, then “The number is 10” is printed on the console; if not, then “The number is not 10” is printed on the console.
3. If a number exists on the number line, then "Number exists" is printed on the console; if not, then "Imaginary number" is printed on the console.
4. All are suitable.

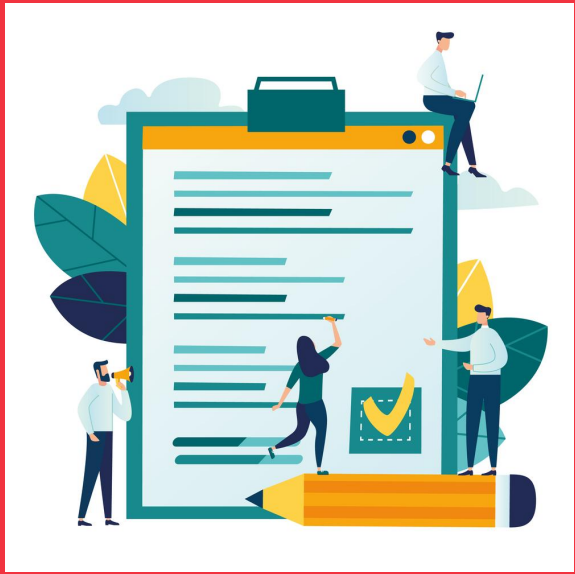
Poll 2 (Answer)

In which of the following scenarios would ‘switch..case’ be unsuitable?

1. If the signal is green, then turn on a machine, and if the signal is red, then turn off the machine.
2. If a number is equal to 10, then “The number is 10” is printed on the console; if not, then “The number is not 10” is printed on the console.
3. **If a number exists on the number line, then "Number exists" is printed on the console; if not, then "Imaginary number" is printed on the console.**
4. All are suitable.

Today's Agenda

- Arrays in JavaScript
- Objects in JavaScript
- Functions in JavaScript

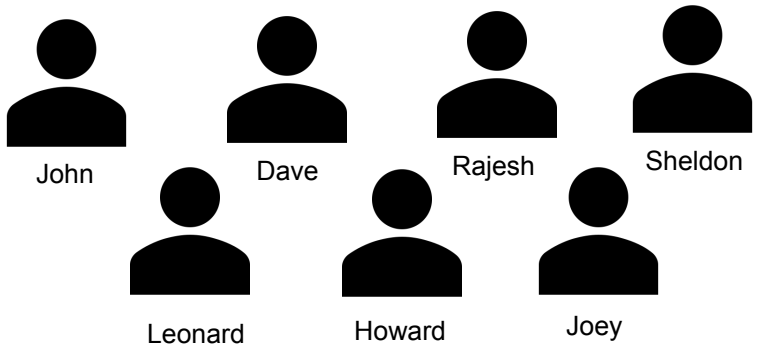


Arrays

Introduction to Arrays

- JavaScript (JS) arrays are used for storing multiple values in a single variable.
- For example, you can individually store the values of the names of students mentioned to the right, such as the following:

```
var student1 = “John”, student2 = “Dave”,
student3 = “Rajesh”;
```



- However, if you have a list of students and need to loop through it, then, instead of having different variables for each student, you will need one variable that stores the data of all the students.
- An array can store many values under a single name, and an individual value can be accessed by its position, which is called an index. The indexing starts from 0. So, the first position is 0, the second position is 1, and so on.

```
var students = [ “John”, “Dave”, “Rajesh”, “Sheldon”, “Leonard”, “Howard”, “Joey” ];
```

JavaScript Syntax

upGrad

Array Creation, Access and Changing Elements

JavaScript (JS) arrays can be created using the **[ ]** notation.

```
var students = [ “John”, “Dave”, “Rajesh”, “Sheldon”, “Leonard”, “Howard”, “Joey” ];
```

To access an element in an array, you can call it using its position called index. The first index is 0, the second index is 1, and so on.

```
console.log(students[0]); //John
console.log(students[3]); //Sheldon
```

You can change an element in an array by calling the index and then supplying it with a new value.

```
students[0] = “Chandler”;
console.log(students); //[Chandler,Dave,Rajesh,Sheldon,Leonard,Howard,Joey]
```

Array properties and methods

The length of the array can be measured using the **length** property. You can also access the last element of the array using this property.

```
var students = [ “John”, “Dave”, “Rajesh”, “Sheldon”, “Leonard”, “Howard”, “Joey” ];
console.log(students.length); //7
console.log(students[students.length - 1]); //Joey
```

To add new elements to an array, you can use **push() method** or the **length** property. However, you should not add elements whose index is much higher than the length of the array, as that will create holes in the array.

```
students.push(“Ross”);
console.log(students); //[John,Dave,Rajesh,Sheldon,Leonard,Howard,Joey,Ross]
students[students.length] = “William”;
console.log(students); //[John,Dave,Rajesh,Sheldon,Leonard,Howard,Joey,Ross,William]
students[12] = “Barney”;
console.log(students);
//[John,Dave,Rajesh,Sheldon,Leonard,Howard,Joey,Ross,William,undefined,undefined,undefined,Barney]
```

JavaScript Syntax

upGrad

Array properties and methods

Use the **pop()** method to remove (pop out) the last element in an array. Logging its value in the console will display the removed element.

```
console.log(students.pop()); // Joey
console.log(students); //[John,Dave,Rajesh,Sheldon,Leonard,Howard]
```

An array can be converted to a string using either the **toString()** method or the **join()** method. In the **join()** method, you can specify a separator.

```
var students = [ “John”, “Dave”, “Rajesh”, “Sheldon”, “Leonard”, “Howard”, “Joey” ];
console.log(students.toString()); //John,Dave,Rajesh,Sheldon,Leonard,Howard,Joey
console.log(students.join(“ / ”)); //Joey / Dave / Rajesh / Sheldon / Leonard / Howard / Joey
```

Use the **delete** method to delete an element from an array. This will replace that value with **undefined**.

```
delete students[0];
console.log(students); // [empty,Dave,Rajesh,Sheldon,Leonard,Howard]
console.log(students[0]); //undefined
```

Array Properties and Methods

An array can be shifted using the **shift()** and **unshift()** methods. The **shift()** method removes the first element from an array and shifts all other elements to the left side with a lower index. Outputting the value of the **shift()** method will show the removed element.

On the other hand, the **unshift()** method adds a new element to the beginning of an array and moves all other elements to the right side with a higher index.

```
var students = [“Rajesh”, “Sheldon”, “Leonard”, “Howard”];
console.log(students.shift()); // Rajesh
console.log(students); // [Sheldon,Leonard,Howard]
console.log(students.unshift(“Stuart”)); //4
console.log(students); // [Stuart,Sheldon,Leonard,Howard]
```

You can merge two or more arrays using the **concat()** method. This method does not change the original arrays but returns a new array. The **concat()** method can take any number of arrays as arguments.

```
var arr1 = [“BMW”, “Audi”, “Mercedes”], arr2 = [“Honda”, “Hyundai”], arr3 = [“Tata”];
var cars = arr1.concat(arr2, arr3); // [BMW,Audi,Mercedes,Honda,Hyundai,Tata]
```

Array Properties and Methods

The **splice()** method allows you to add new items to an array or remove/replace existing items in an array. The first parameter in the splice() method defines the *position where new elements need to be added*, the second parameter defines *how many elements need to be removed*, and the remaining parameters define the new elements to be added. If only the first two parameters are specified, then the elements from an array without converting their positions to **undefined** are removed. Outputting the value of the splice() method will return the removed elements.

```
var students = [ “Rajesh”, “Sheldon”, “Leonard”, “Howard”];
console.log(students.splice(2, 2, “Joey”, “Chandler”)); // [Leonard,Howard]
console.log(students); // [Rajesh,Sheldon,Joey,Chandler]
console.log(students.splice(1, 2)); // [Sheldon,Joey]
console.log(students); // [Rajesh,Chandler]
```

The **slice()** method slices a piece of an array into a new array. If only one argument in the parameters is specified, then it slices the array from that position to the end of the array. If two parameters are included, then it slices the array from the position specified by the first argument to the position specified by the second argument. However, this method does not include the position specified by the second argument. Note that the original array is not modified.

```
var students = [“Rajesh”, “Sheldon”, “Leonard”, “Howard”, “Joey”, “Chandler”, “Ross”];
console.log(students.slice(2)); // [Leonard,Howard,Joey,Chandler,Ross]
console.log(students); // [Rajesh,Sheldon,Leonard,Howard,Joey,Chandler,Ross]
console.log(students.slice(1, 3)); // [Sheldon,Leonard]
```

Poll 3 (15 Sec)

What will be the output of the following program?

```
var array = ["Mountains","Sea","Island","Forest","Cave"];
array.splice(2);
console.log(array);
```

1. ["Mountains", "Sea"]
2. ["Island","Forest","Cave"]
3. ["Mountains","Sea","Island","Forest","Cave"]
4. No output

Poll 3 (Answer)

What will be the output of the following program?

```
var array = ["Mountains","Sea","Island","Forest","Cave"];
array.splice(2);
console.log(array);
```

1. **["Mountains", "Sea"]**
2. ["Island","Forest","Cave"]
3. ["Mountains","Sea","Island","Forest","Cave"]
4. No output

Hands-On Exercise (3 min)

upGrad

Suppose you are in Washington for some work and are planning to go on vacation to another city. However, the catch is that with the money that you currently have, you can only travel to a city that is within 4,000 miles from Washington. Here is the list of cities along with their distance from Washington (arranged in ascending order):
Hamilton: 334 miles, Brussels: 4,500 miles, Toronto: 353 miles, Sydney: 9,600 miles, London: 3,500 miles, Vancouver: 2,300 miles

Here is an array with these cities arranged in ascending order with respect to their distance from Washington:

```
var cities = ['Hamilton', 'Toronto', 'Vancouver', 'London', 'Brussels', 'Sydney']
```

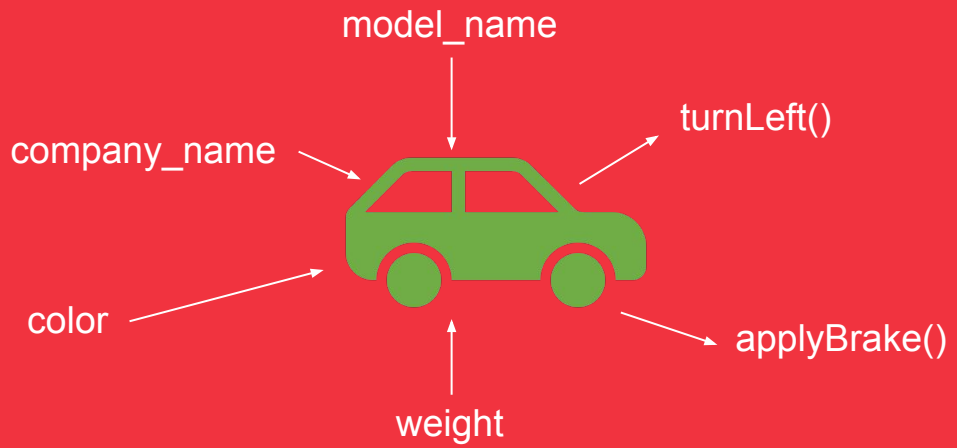
Write single-line array operations (one after the other) that:
1. Remove the cities to which you cannot go;
2. Add a new city 'Tokyo' at the end of the list, which is the farthest from Washington (10,000 miles);
3. Add a new city 'Boston' at the beginning of the list, as it is nearest to Washington (250 miles); and
4. Remove the farthest city from the list (Use splice() method)

Note that each operation mentioned above must be performed in a single line and one after the other. After you have performed all these steps, you have to print the original array.

Stub Code: Click here.

Solution: Click here.

JavaScript Objects



JavaScript Objects

upGrad

Introduction to Objects

- JavaScript (JS) objects are similar to real-life objects. For instance, a car can be an object.
- Objects, such as a car, have properties and methods.
- Properties of a car include the car name, the model number as well as the color and weight of the car, whereas methods include starting a car, turning left, turning right and applying brakes.

All cars have the same properties, but their values differ. Similarly, all cars have the same methods, but they are performed at different times.

```
var car = { name: “Mercedes”, model: “C200”, color: “white”, weight: 500 };
```

Objects are variables that can have multiple values. They are denoted by {}. Objects can contain multiple properties, and every property must be in a **key: value** pair.

JavaScript Objects

upGrad

Object methods

```
var person = {
    firstName: “John”,
    lastName: “Doe”,
    age: 27,
    getFullName: function() {
       return this.firstName + “ ” + this.lastName;
    }
};
```

- JavaScript (JS) objects can also have methods. For instance, the ***getFullName*** property is a method inside the object that returns the full name by concatenating the first name and the last name inside the object.
- In programming, the ***this*** keyword refers to the owner of a property. In the case given above, the property ***fullName*** is owned by the object ***person***. Therefore, ***this*** is equal to a *person* object.
- Therefore, ***this.firstName*** will give the output John, whereas ***this.lastName*** will give the output Doe.

JavaScript Objects

upGrad

Accessing an object

An object can have another object inside it, which is called a nested object. In the example given below, *address* is another object nested inside the object *person*.

```
var person = {
    firstName: “John”,
    lastName: “Doe”,
    age: 27,
    getFullName: function() {
       return this.firstName + “ ” + this.lastName;
    },
    address: {
         building: “Villa Apartments”,
         road: “Rue La Roi”,
         city: “Paris”
    }
};
```

JavaScript (JS) objects can be accessed in two ways: **bracket notation** and **dot notation**.

```
console.log(person[“firstName”]); //John

console.log(person.firstName); //John

console.log(person.getFullName()); //John Doe

console.log(person.address.city); //Paris
```

Accessing Objects

You can also loop through the different properties of the objects using the for...in loop.

```
var person = {
    firstName: “John”,
    lastName: “Doe”,
    age: 27,
    getFullName: function() {
       return this.firstName + “ ” + this.lastName;
    },
    address: {
         building: “Villa Apartments”,
         road: “Rue La Roi”,
         city: “Paris”
    }
};
```

```
var prop;
for (prop in person) {
  console.log(prop);
}

/* OUTPUT:
firstName
lastName
age
getFullName
address
*/
```

Poll 4 (15 Sec)

Which of the following is the correct way of accessing the field "id" of the object given in the adjoining screenshot?
(Note: More than one option may be correct.)

1. person[id]
2. person["id"]
3. person.id
4. person."id"

```
var person = {
    id: 101,
    email: 'alice11@gmail.com',
    personalInfo: {
          name: 'Alice',
          address: {
              line1: 'Smith Avenue',
          }
     }
};
```

Poll 4 (Answer)

Which of the following is the correct way of accessing the field "id" of the object given in the adjoining screenshot?
(Note: More than one option may be correct.)

1. person[id]
2. **person["id"]**
3. **person.id**
4. person."id"

```
var person = {
    id: 101,
    email: 'alice11@gmail.com',
    personalInfo: {
          name: 'Alice',
          address: {
              line1: 'Smith Avenue',
          }
     }
};
```

Poll 5 (15 Sec)

What will be the output of the program given in the adjoining screenshot?

1. guitar
2. violin
3. rock
4. classical

```
var music = {
    rock: 'guitar', classical: 'violin'
};
var classical = 'rock';
var sound = music[classical];
console.log(sound);
```

Poll 5 (Answer)

What will be the output of the program given in the adjoining screenshot?

```
var music = {
    rock: 'guitar', classical: 'violin'
};
var classical = 'rock';
var sound = music[classical];
console.log(sound);
```

1. **guitar**

You are trying to access the "classical" property of the music object. However, the variable “classical” has itself been changed to "rock". Thus, when you try to access it, it gives us the value corresponding to "rock", that is, "guitar".

1. violin
2. rock
3. classical

JavaScript Objects

upGrad

Adding and deleting properties of an object

- You cannot add a property to an object that does not exist.
- To add a property to an existing object, you simply need to declare it using the standard bracket or dot notation and then assign a value to it.
- Similarly, to delete an object's property, you can use the **delete** keyword.

```
var person = {
    firstName: "John",
    lastName: "Doe",
    age: 27,
};

person["nationality"] = "British";
person.eyeColor = "blue";

console.log(person.nationality); //British

delete person.age;
console.log(person); // {firstName: "John", lastName: "Doe"}
```

JavaScript Objects

upGrad

Looping over an object

- You can loop through an object using for...in loop.

```
var person = {
    firstName: “John”,
    lastName: “Doe”,
    age: 27,
};
var i,details="";
for (i in person) {
  details += person[i] + " ";
}
// "John Doe 27 "
```

Poll 6 (15 Sec)

Which of the following statements about objects is FALSE?

1. Values in the objects can be accessed in a way that is similar to how an element in an array is accessed. So, the second value can be accessed by writing object[2].
2. Objects can store values of multiple data types.
3. Objects store data using key-value pairs.
4. You can iterate over an object using a for...in loop.

Poll 6 (Answer)

Which of the following statements about objects is FALSE?

1. **Values in the objects can be accessed in a way that is similar to how an element in an array is accessed. So, the second value can be accessed by doing object[2].**
2. Objects can store values of multiple data types.
3. Objects store data using key-value pairs.
4. You can iterate over an object using a for...in loop.

JavaScript Objects

upGrad

JavaScript Constructors

Sometimes, you need to create a blueprint for creating many objects of the same type. For instance, in the example given below, three different people possess the same properties such as the first name, last name, age and eye-color. So, instead of creating a new object every time with new values, you can create a function that can be instantiated with the values. This function is called a **constructor**.

firstName	John
lastName	Doe
age	27
eyeColor	blue

firstName	Jane
lastName	Dias
age	25
eyeColor	brown

firstName	Rachel
lastName	Greene
age	30
eyeColor	black

```
function Person (first, last, age, color) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = color;
}
```

```
var myDad = new Person("John", "Doe", 27, "blue");
var myMom = new Person("Jane", "Dias", 25, "brown");
var mySister = new Person("Rachel", "Greene", 30, "black");
console.log(myDad.firstName, myMom.firstName,
mySister.firstName); // John Jane Rachel
```

JavaScript Objects

upGrad

JavaScript Prototype

You cannot add a property to the JavaScript constructor in a way that is similar to the one in which you add a property to an object. You use the **prototype** property to add a property or a method to a constructor so that it is available to all the objects created using the constructor.

```
function Person (first, last, age, color) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = color;
}
```

```
Person.prototype.language = “English”;
```

```
Person.prototype.getFullName = function() {
     return this.firstName + “ ” + this.lastName;
}
```

```
var myDad = new Person("John", "Doe", 27,
"blue");

console.log(myDad.language); // English
console.log(myDad.getFullName()); // John
Doe
```

JavaScript Objects

upGrad

Pass by Reference

In JavaScript, objects are passed as reference and not as values (i.e., not copied). Two objects that point towards each other will always remain identical, and changes in one object will automatically reflect in the other.

Note that this holds true only for user-defined types (objects & arrays) and not primitive data types such as strings or numbers.

```
// consider two objects
var a = [1,2,3];
var b = a;
//At this stage, both a and b point to each other and have the value [1,2,3].
//Now when we change the contents of a, the contents of b change automatically.
//Suppose, we change the value of a[0] to 99.
a[0] = 99;
/*Now when we try to print the value of b[0], which has not been explicitly re-assigned, it gives us the value as 99, and not 1. This so happens because the variable b is still pointing to the original array, in which a[0] now holds the value of 99. Hence, it can be concluded that in Javascript, non-primitive types (objects & arrays) are passed as reference and not as values.*/
console.log(b[0]);
```

upGrad

Poll 7 (15 Sec)

What is the output of the code snippet given in the adjoining screenshot?

1. 11
2. [11, 20, 99]

1. [11,22,33]
2. Error

```
var a = [11,22,33];
var b = a;
a[0] = [11,20,99];
console.log(b[0]);
```

Poll 7 (Answer)

What is the output of the code snippet given in the adjoining screenshot?

1. 11
2. **[11, 20, 99]**
3. [11,22,33]
4. Error

```
var a = [11,22,33];
var b = a;
a[0] = [11,20,99];
console.log(b[0]);
```

Poll 8 (15 Sec)

What is the output of the code snippet given in the adjoining screenshot?

```
var a = 10;
var b = a;
a = 15;
console.log(a, b);
```

1. 10 15
2. 15 10
3. 15 15
4. None of the above

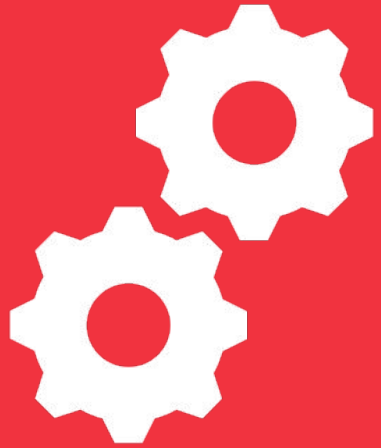
Poll 8 (Answer)

What is the output of the code snippet given in the adjoining screenshot?

```
var a = 10;
var b = a;
a = 15;
console.log(a, b);
```

1. 10 15
2. **15 10**
3. 15 15
4. None of the above

JavaScript Functions



Introduction to Functions and Declaring a Function

- In simple terms, JavaScript (JS) function is a block of code designed to perform a specific task.
- A JS function is **executed** when it is **invoked (or called)**.

The syntax of JS functions is as follows:

```
function functionName (argument_1, argument_2,..., argument_n) {
   //code to be executed
}
```

A JavaScript function is defined with a **function** keyword, followed by a name for the function and the parentheses **()**.

Function names follow the same rule as variables. Inside the parentheses, you can include several parameter that can be accessed in the body of the function inside curly braces. These parameters become the variables inside the function body. The values provided as argument while invoking the function are assigned to these parameters in the order.

```
function add (number1, number2) {
   return number1 + number2;
}
```

JavaScript Functions

upGrad

Calling a Function

A JavaScript (JS) function executes the block of code inside it only when it is invoked (or called).

The syntax for invoking (or calling) a function is as follows:

```
functionName(argument_1, argument_2,..., argument_n)
```

Calling a function involves the function name followed by parentheses and the arguments inside the parentheses. These arguments are passed to the parameters defined in the function definition.

```
function add(x, y) {
   alert("Addition");
   var sum = x + y;
   return sum;
}
var result = add(2, 4);
```

In the example given above, a function named **sayHi** has no arguments, and it simply brings an alert box with the message "Hello World". So, when the function is called using the statement **sayHi()**, it will execute the code, and an alert box will be shown with the text "Hello World".

JavaScript Functions

upGrad

Keyword Argument

- The **argument** keyword is a local variable available for functions (except ES6 functions, which will be covered later).
- You can refer to a function's arguments using the **argument** keyword, which is an object type.
- All the arguments are indexed, and indexing starts from 0.
- For instance, in the example given below, you will call argument[0] to access the first argument. Similarly, the second argument will be called using argument[1].

```
function add(number1, number2, number3) {
   console.log(argument[0]);  //2
   console.log(argument[1]);  //3
   console.log(argument[2]);  //5
}
add(2, 3, 5);
```

JavaScript Functions

upGrad

Different ways of defining a function

Named function: This is the function declaration and the function invocation (calling) that you witnessed before.

```
function add(number1, number2) {
   return number1 + number2;
}
add(2, 3);
```

JavaScript Functions

upGrad

Different ways of defining a function

Immediately invoked function: It is a way of executing functions immediately after they are created. This process of function definition does not pollute global objects.

```
(function(number1, number2) {
return number1 + number2;
})(1, 2);
```

JavaScript Functions

upGrad

Different ways of defining a function

Anonymous function: This process of defining a function involves creating a function without giving it a name. It is generally used as an argument for other functions. For instance, the JavaScript method of setting a timer *setTimeout()* uses anonymous functions.

```
var sum = function(x, y) {
return x + y;
}
```

```
setTimeout(function () {
  console.log('Time out of 3s')
}, 3000);
```

You can read more about it here.

Different ways of defining a function

Recursive function: A recursive function is a function that calls itself until a condition to call itself fails. For instance, a countdown timer function will call itself until the number becomes 0.

```
function countdown(num) {
   console.log(num);
   var nextNum = num - 1;
   if(nextNum > 0) { countdown(nextNum); }
}
countdown(5);
```

JavaScript Functions

upGrad

Method

Have you ever wondered whether or not you can put a function inside an object?

Do you remember seeing a similar type of code while studying objects?

```
var person = {
    firstName: “Prachi”,
    lastName: “Agrawal”,
    getFullName: function() {
       console.log(this.firstName + “ ” + this.lastName);
    }
};
```

Here, the ***function*** is called a ***method***.

Thus, ***a method is a function that is assigned to a key in an object.***

```
person.getFullName();
```

Poll 9 (15 Sec)

What is the return type of the function given in the adjoining screenshot?

1. undefined
2. string
3. number
4. function

```
function f(x) {
    return x * x;
};
```

Poll 9 (Answer)

What is the return type of the function given in the adjoining screenshot?

1. **undefined**
2. string
3. number
4. function

```
function f(x) {
    return x * x;
};
```

Poll 10 (15 Sec)

What will be printed on the console after running the code given in the adjoining screenshot?

1. Dan Brown is famous for his fictional novels.
2. function() { console.log("Dan Brown is famous for his fictional novels.") }
3. Dan Brown
4. undefined

```
var author = {
    name: "Dan Brown",
    getBook: function() {
        console.log("Dan Brown is famous for his fictional novels.");
    }
}
author.getBook();
```

Poll 10 (Answer)

What will be printed on the console after running the code given in the adjoining screenshot?

1. **Dan Brown is famous for his fictional novels.**
2. function() { console.log("Dan Brown is famous for his fictional novels.") }
3. Dan Brown
4. undefined

```
var author = {
    name: "Dan Brown",
    getBook: function() {
        console.log("Dan Brown
is famous for his fictional
novels.");
    }
}
author.getBook();
```

Hands-On Exercise (3 mins)

upGrad

Write a function that calculates the result of a base raised to the power of an exponent, and name this function power.

Sample Input:
base = 2, exponent = 3

Sample Output:
8

The stub code is provided here.

The solution is provided here.

Tasks to complete after today's session

MCQs
Coding Questions

Key Takeaways

upGrad

- JavaScript (JS) objects are similar to real-life objects (e.g., car) and have properties (e.g., doors, colours and wheels) and methods (e.g., turn left, turn right and apply brakes). All the properties of and methods in objects are stored in the “key-value” format.

- User-defined types (objects and arrays) are passed by reference whereas the primitive types are passed by value.

- The functions in JS are blocks of code that execute a particular task, such as adding two numbers. These functions may or may not return a value.

In the next class, we will discuss...

- Scope
- Closure
- Hoisting

upGrad

#RahoAmbitious

Thank you!