upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: JavaScript and Server-Side Communication

Lecture On : JS Methods and Introduction to DOM

Instructor : Siddhesh Prabhugaonkar

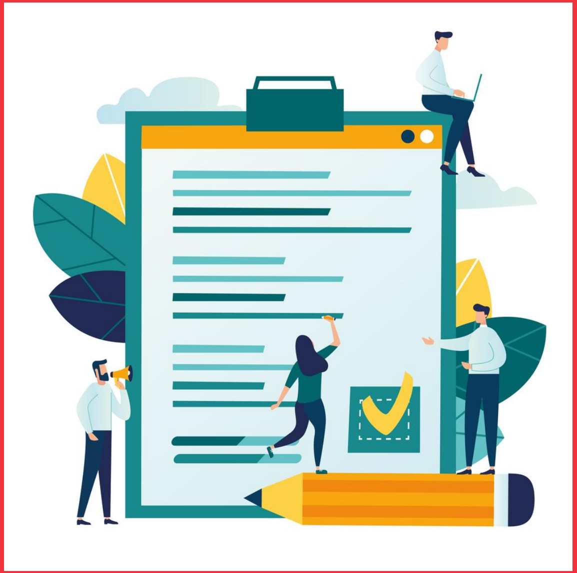
upGrad

In the previous session, we covered….

- Scope
- Closure
- Hoisting

Today’s Agenda

- ‘this’ keyword
- bind(), call(), and apply()
- Introduction to DOM manipulation



Window Object

upGrad

- The window object represent the window of your browser.
- The window object is the owner of all the global properties of JavaScript global variables and objects.
- All the global functions are the methods of the window object.

‘this’ keyword

‘this’ keyword

- The JavaScript **this** keyword refers to the object it belongs to. For different scenarios, it has a different meaning.
- In a method of an object, **this** refers to the owner object.

```
<!DOCTYPE html>
<html>
    <body>
        <p>Here, this keyword represents the car object.</p>
        <script>
            var car = {
                carName: "i20",
                carCompany : "Hyundai",
                carDetails : function() {
                    return this.carName + " " + this.carCompany;
                }
            };
            console.log(car.carDetails());
        </script>
    </body>
</html>
```

‘this’ keyword

upGrad

- When **this** is used outside a method/function, the owner is the global object (i.e. the window object).

```
<!DOCTYPE html>
<html>
    <body>
        <p>When this keyword is used alone</p>
        <script>
            var car = this;
            console.log(car); //[object Window]
        </script>
    </body>
</html>
```

‘this’ keyword

upGrad

- In a function, **this** refers to the owner of the function which is global object [object Window].

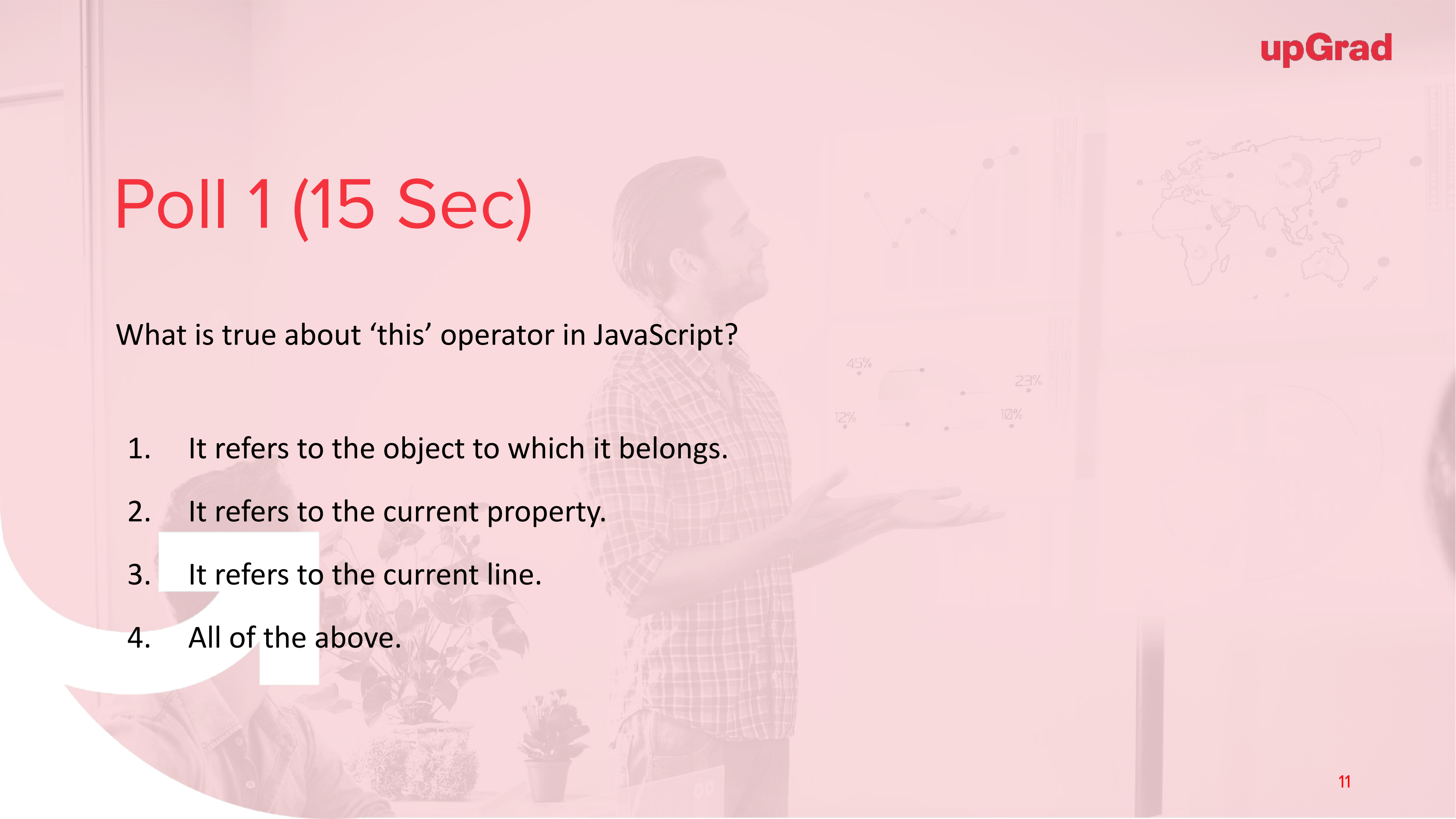
```
<!DOCTYPE html>
<html>
    <body>
        <p>Here, this keyword is used inside a function</p>
        <script>
            function func() {
                return this;
            }
            console.log(func());
        </script>
    </body>
</html>
```

‘this’ keyword

upGrad

- For event handlers, **this** refers to the element that received the event.

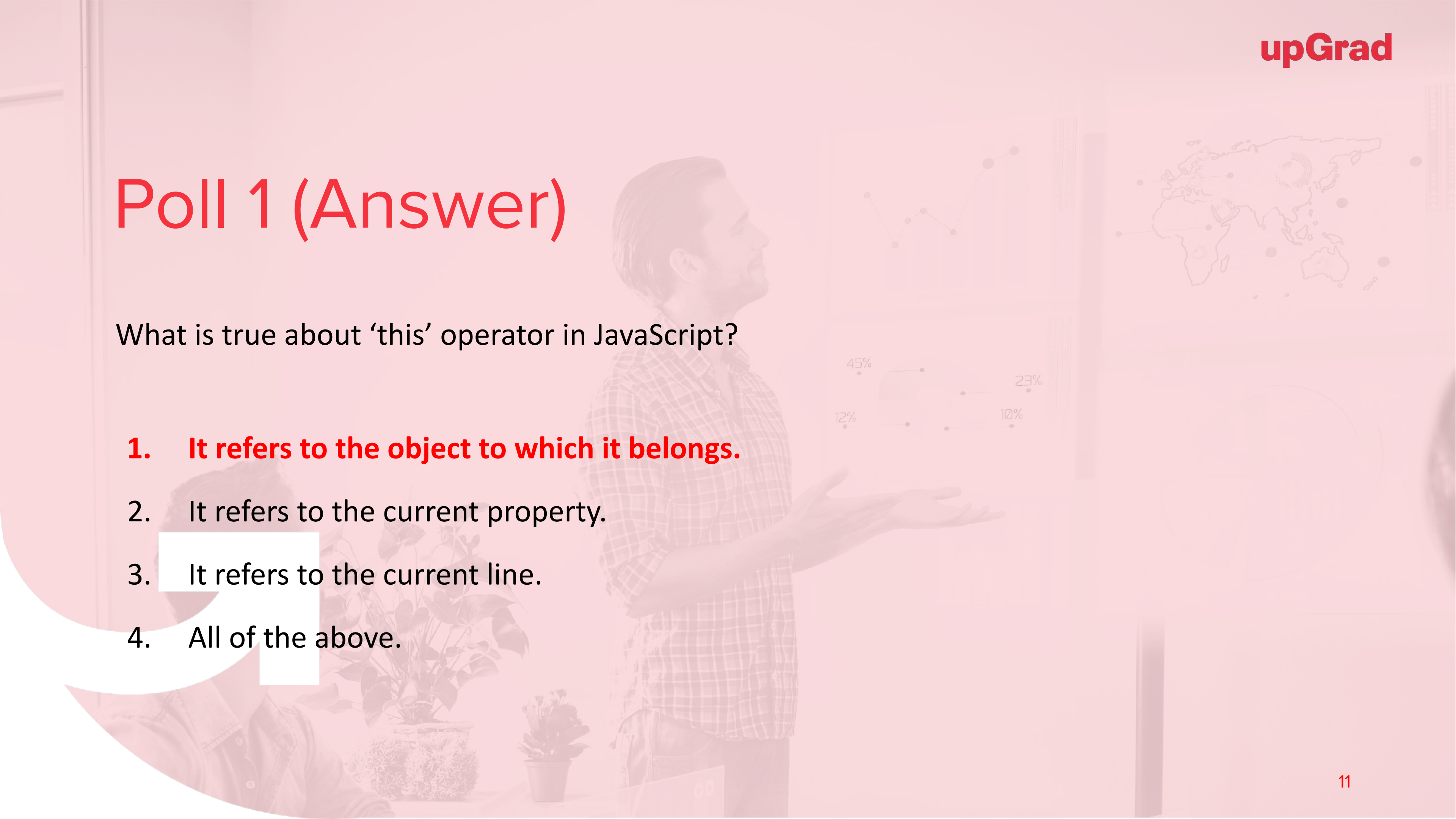
```html
<!DOCTYPE html>
<html>
    <body>
        <button onclick="this.style.color='green'">Submit</button>
    </body>
</html>
```



Poll 1 (15 Sec)

What is true about ‘this’ operator in JavaScript?

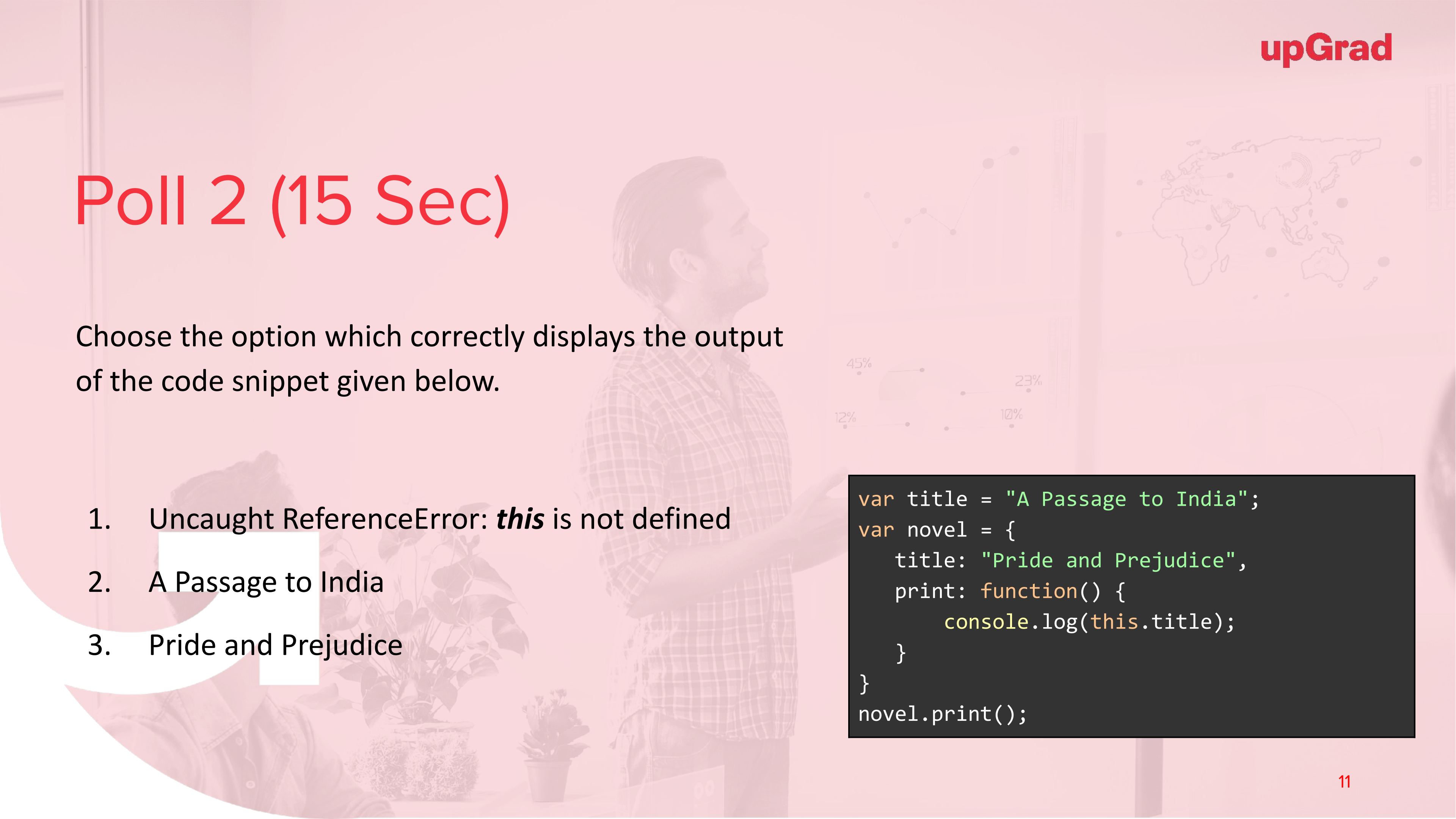
1. It refers to the object to which it belongs.
2. It refers to the current property.
3. It refers to the current line.
4. All of the above.



Poll 1 (Answer)

What is true about ‘this’ operator in JavaScript?

1. **It refers to the object to which it belongs.**
2. It refers to the current property.
3. It refers to the current line.
4. All of the above.



Poll 2 (15 Sec)

Choose the option which correctly displays the output of the code snippet given below.

1. Uncaught ReferenceError: ***this*** is not defined
2. A Passage to India
3. Pride and Prejudice

```
var title = "A Passage to India";
var novel = {
   title: "Pride and Prejudice",
   print: function() {
       console.log(this.title);
   }
}
novel.print();
```

upGrad

Poll 2 (Answer)

Choose the option which correctly displays the output of the code snippet given below:

1. Uncaught ReferenceError: ***this*** is not defined
2. A Passage to India
3. **Pride and Prejudice**

```
var title = "A Passage to India";
var novel = {
   title: "Pride and Prejudice",
   print: function() {
       console.log(this.title);
   }
}
novel.print();
```

setTimeout() method

upGrad

- The setTimeout() method is used to a timer for the execution of a specific piece of code.
- The syntax of the setTimeout() method is:

```
var timeoutID = scope.setTimeout(function[, delay, arg1, arg2, ...]);
var timeoutID = scope.setTimeout(function[, delay]);
var timeoutID = scope.setTimeout(code[, delay]);
```

For example:

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

setTimeout(person.getFullName, 1000);
```

bind(), call(), and apply()

bind(), call(), and apply()

upGrad

bind()

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

setTimeout(person.getFullName, 1000); //undefined undefined
```

- In the code shown above, **setTimeout** results in undefined. It is because when you use setTimeout, as per JavaScript principles, the **this** inside the setTimeout is set to the global object.
- This is the reason why **this** set inside the fullName method is not associated with the person object.

bind(), call(), and apply()

upGrad

bind()

- The **bind()** method returns a new function, which when called, has its **this** set to a specific value.

- For instance, in the example below, the *bind()* method binds the *person.getFullName()* method to the *person* object. This is how the **'this'** value is restored to the person object.

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

var greet = person.getFullName.bind(person);
setTimeout(greet, 1000); //John Doe
```

Poll 3 (15 Sec)

What is the output?

1. Prachi Agrawal
2. Prachi
3. Agrawal
4. Error

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

var greet = person.getFullName.bind(person);
setTimeout(greet, 5000);
```

Poll 3 (Answer)

What is the output?

1. Prachi Agrawal
2. **Prachi**
3. Agrawal
4. Error

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

var greet = person.getFullName.bind(person);
setTimeout(greet, 5000);
```

bind(), call(), and apply()

call()

- With the JavaScript **call()** method, you can write a method that can be used on different objects.

```
var person = {
    getFullName: function() {
      console.log(this.firstName + " " + this.lastName);
    }
};
var person1 = { firstName: “John”, lastName: “Doe” };
var person2 = { firstName: “Jane”, lastName: “Dias” };
person.getFullName.call(person1); //John Doe
person.getFullName.call(person1); //Jane Dias
```

```
var person = {
    getFullName: function(age, nationality) {
       return this.firstName + “ ” + this.lastName + “, ” + age +
“, ” + nationality;
    }
};
var person1 = { firstName: “John”, lastName: “Doe” };
person.getFullName.call(person1, 27, “Swedish”); //John Doe, 27,
Swedish
```

- In the final line, we call the person’s *getFullName()* method on the object *person1*.
- This passes the *firstName* and the *lastName* of *person1* to the *person* object.
- Similarly, in the second example, we can also call the *getFullName()* method with arguments.
- We have to pass arguments as **strings**, separated by commas.
- The function on which you apply the call() method is invoked implicitly when the control reaches it. Thus, you do not need to explicitly call the function on which you apply the call() method.

Poll 4 (15 Sec)

What is the output?

1. Outer: Prachi
   Developer
   Inner: Prachi

2. Outer: Prachi
   Inner: Prachi
   Developer

```
var person = {
   firstName: "Prachi",
   get: function() {
      console.log("Outer: " + this.firstName);
      var print = function(role) {
         console.log(role);
         console.log("Inner: " + this.firstName);
      }.call(this, "Developer");
   }
};
person.get();
```

Poll 4 (Answer)

What is the output?

1. **Outer: Prachi**
   Developer
   Inner: Prachi

2. Outer: Prachi
   Inner: Prachi
   Developer

```javascript
var person = {
   firstName: "Prachi",
   get: function() {
      console.log("Outer: " + this.firstName);
      var print = function(role) {
         console.log(role);
         console.log("Inner: " + this.firstName);
      }.call(this, "Developer");
   }
};
person.get();
```

‘this’, bind(), call(), and apply()

upGrad

apply()

- The **apply()** method is similar to the call() method. They are also called in the similar way; the only difference is that when you are using **apply()** with arguments, you must supply arguments as an array.

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

var person1 = { firstName: “John”, lastName: “Doe” };
person.getFullName.apply(person1); //John Doe
```

```
var person = {
    getFullName: function(age, nationality) {
        return this.firstName + “ ” + this.lastName + “, ” + age + “, ” +
nationality;
    }
};
var person1 = { firstName: “John”, lastName: “Doe” };
person.getFullName.apply(person1, [27, “Swedish”]); //John Doe, 27, Swedish
```

Poll 5 (15 Sec)

What is the output?

1. Outer: Prachi
   Developer
   Inner: Prachi

2. Outer: Prachi
   Inner: Prachi
   Developer

```
var person = {
   firstName: "Prachi",
   get: function() {
      console.log("Outer: " + this.firstName);
      var print = function(role) {
         console.log(role);
         console.log("Inner: " + this.firstName);
      }.apply(this, ["Developer"]);
   }
};
person.get();
```

Poll 5 (Answer)

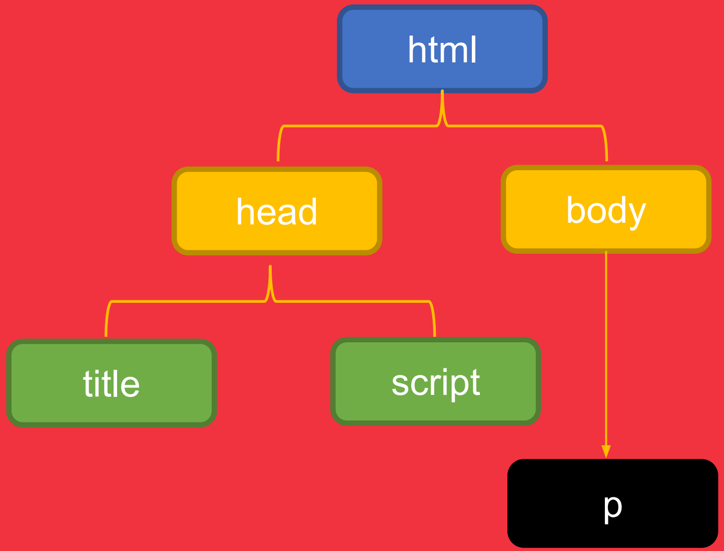
What is the output?

1. **Outer: Prachi**
   Developer
   Inner: Prachi

2. Outer: Prachi
   Inner: Prachi
   Developer

```
var person = {
   firstName: "Prachi",
   get: function() {
      console.log("Outer: " + this.firstName);
      var print = function(role) {
         console.log(role);
         console.log("Inner: " + this.firstName);
      }.apply(this, ["Developer"]);
   }
};
person.get();
```

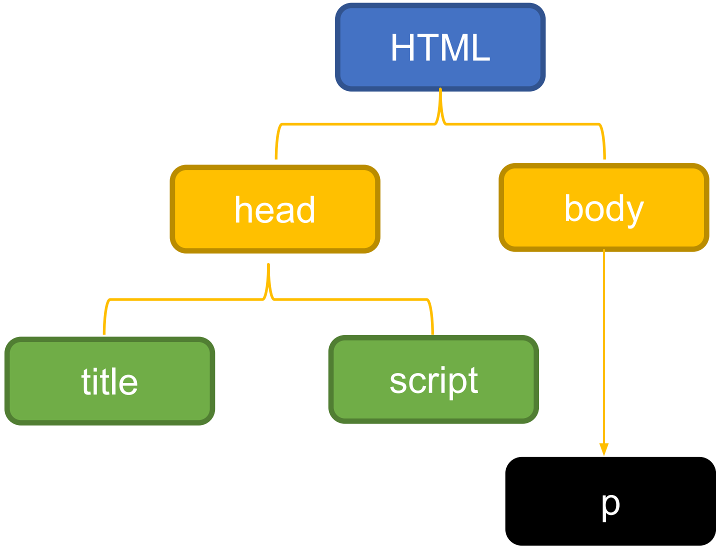
DOM Manipulation



DOM Manipulation

Introduction to DOM

- DOM stands for Document Object Model.
- The HTML DOM model, in a basic sense, is a tree of objects of the HTML elements.
- It defines the HTML elements as objects.
- It also defines the properties of all elements, the methods to access these elements and the events associated with each element.
- With the help of JavaScript, you can access any element on DOM. In JavaScript, you can know the parent element or the children elements of the accessed element. It can also access the properties or methods of that element.
- Using JavaScript, you can also manipulate the accessed elements.



upGrad

DOM Selectors

ID Selector

Class Selector

Tag Selector

DOM Selectors

upGrad

- DOM elements can be accessed in three ways: by using ID, class and the tag name.
- The getElementById property returns the element, getElementsByClassName property returns a collection of all the elements with the class name mentioned in the parentheses, and getElementsByTagName property returns a collection of all the elements with the tag name.

```html
<h1 id="header">World War 2</h1>

<p class="text1">The World War 2 began
in 1939 when Germany declared war on
Poland, followed by England and France
declaring on Germany.</p>

<span>This is a span text.</span>
```

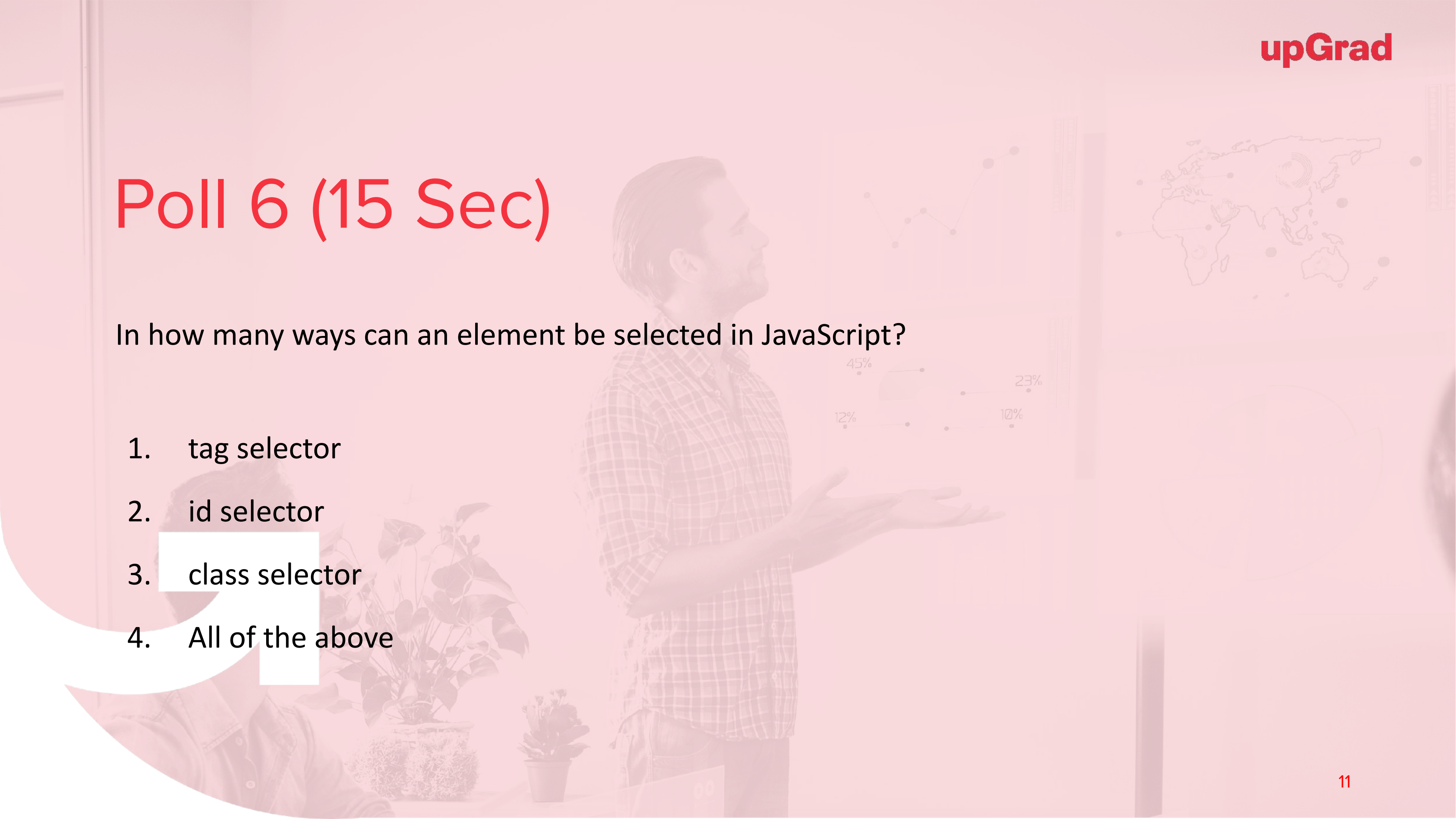
```javascript
//this gets the inner contents of HTML from the element
let text1 = document.getElementById("header").innerHTML;
let text2 = document.getElementsByClassName("text1")[0].innerHTML;
let text3 = document.getElementsByTagName("span")[0].innerHTML;
console.log(text1);
console.log(text2);
console.log(text3);
// this sets the inner contents of HTML to the element
document.getElementById("header").innerHTML = "Second World War";
```

DOM Selectors

upGrad

Once you get the element and store it in a variable, you can view or change the element in the following ways: by viewing/changing the attribute value of the HTML element using setAttribute method.

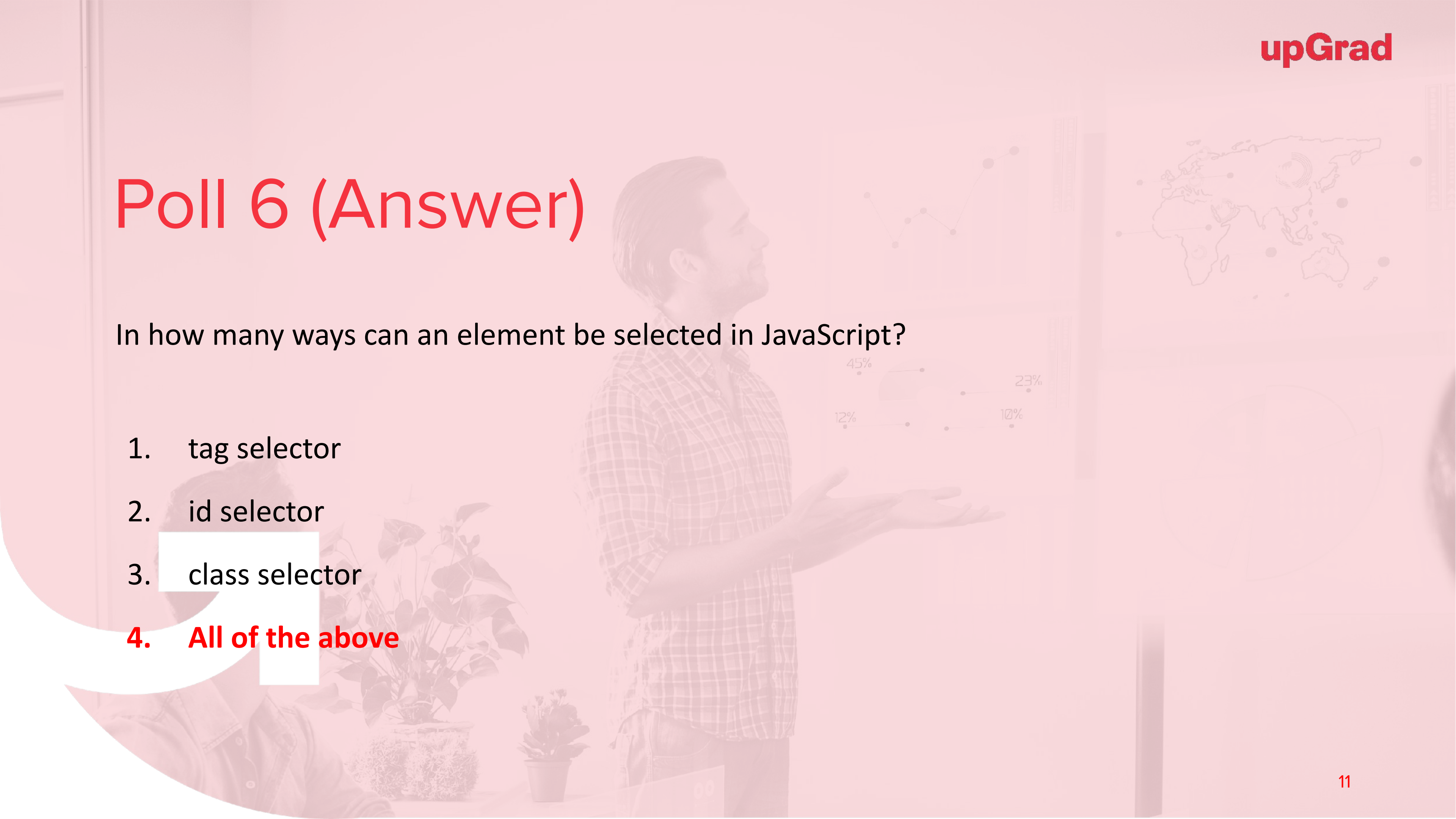
```
<body>
  <h1 id="header">World War 2</h1>
  <img id="image"
src="https://cdn.pixabay.com/photo/2020/07/14/17/04/july-5404922_960_720.jpg"/>
    <script>
      // this sets the attribute of the HTML to the element
      document.getElementById("image").setAttribute("height", "500");
    </script>
</body>
```



Poll 6 (15 Sec)

In how many ways can an element be selected in JavaScript?

1. tag selector
2. id selector
3. class selector
4. All of the above



Poll 6 (Answer)

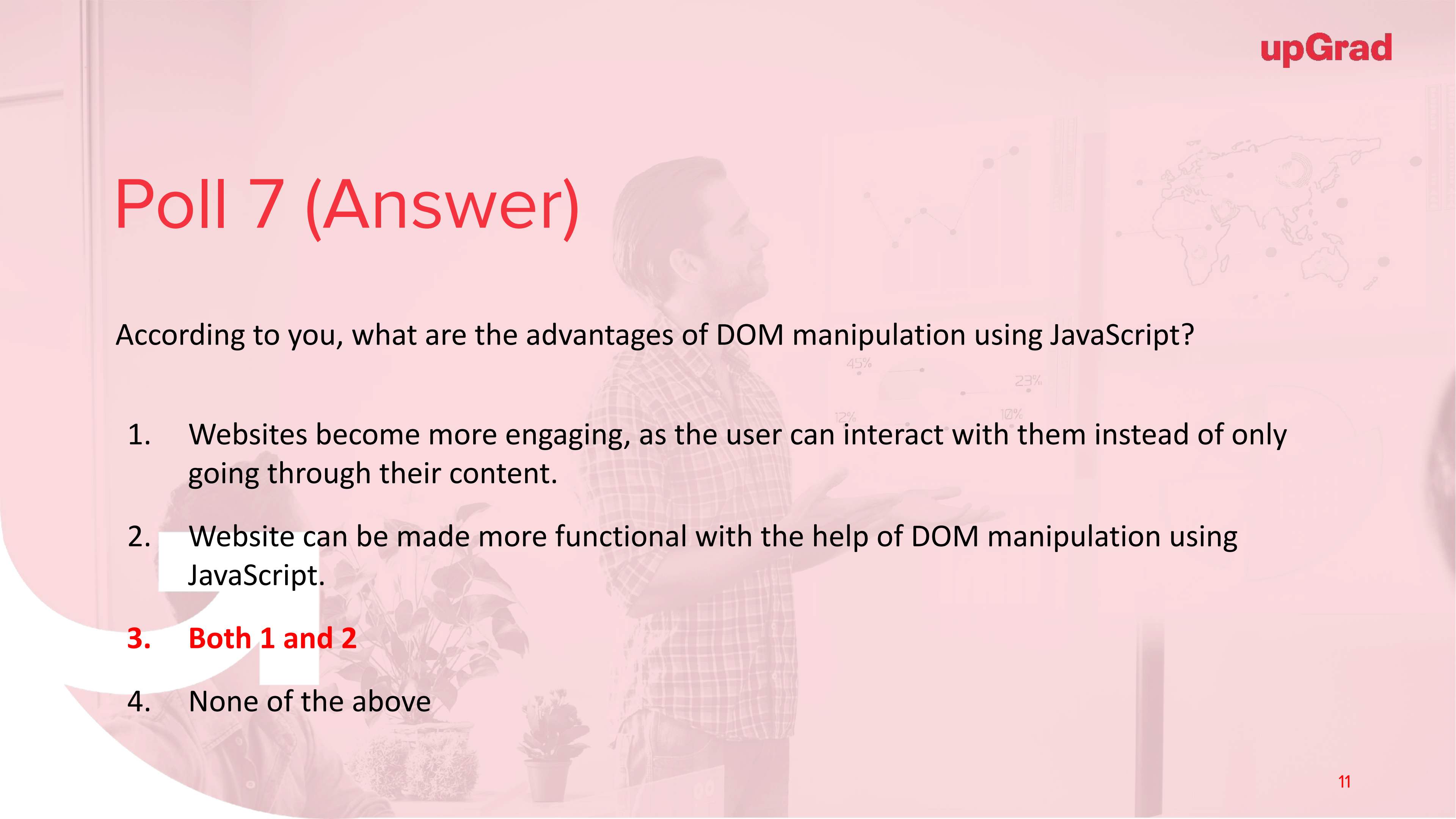
In how many ways can an element be selected in JavaScript?

1. tag selector
2. id selector
3. class selector
4. **All of the above**

Poll 7 (15 Sec)

According to you, what are the advantages of DOM manipulation using JavaScript?

1. Websites become more engaging, as the user can interact with them instead of only going through their content.
2. Website can be made more functional with the help of DOM manipulation using JavaScript.
3. Both 1 and 2
4. None of the above



Poll 7 (Answer)

According to you, what are the advantages of DOM manipulation using JavaScript?

1. Websites become more engaging, as the user can interact with them instead of only going through their content.
2. Website can be made more functional with the help of DOM manipulation using JavaScript.
3. **Both 1 and 2**
4. None of the above

upGrad

Project Work

(Let’s add the required DOM manipulations to our project.)

You can refer to the solution here.

Key Takeaways

- The **this** keyword refers to the owner of the object. The method **bind()** is used for creating a function to return the value of **this** to a function; the method **call()** is used for calling a function with or without the parameters; the method **apply()** is similar to call(), but it accepts parameters as an array, while call() accepts it as a string.

- The difference between bind and call iis that, the bind needs to be explicitly called but the call method is invoked as soon as the control reaches the definition.

Doubts Clearance (5 mins)

upGrad

Task to complete after today’s session

MCQs
Coding Questions
Project - Checkpoint 1

upGrad

In the next class, we will discuss...

- DOM manipulation
- Web Storage

upGrad

#RahoAmbitious

Thank You!