# JAVA SCRIPT

(Scripting Language )

StudentTribe

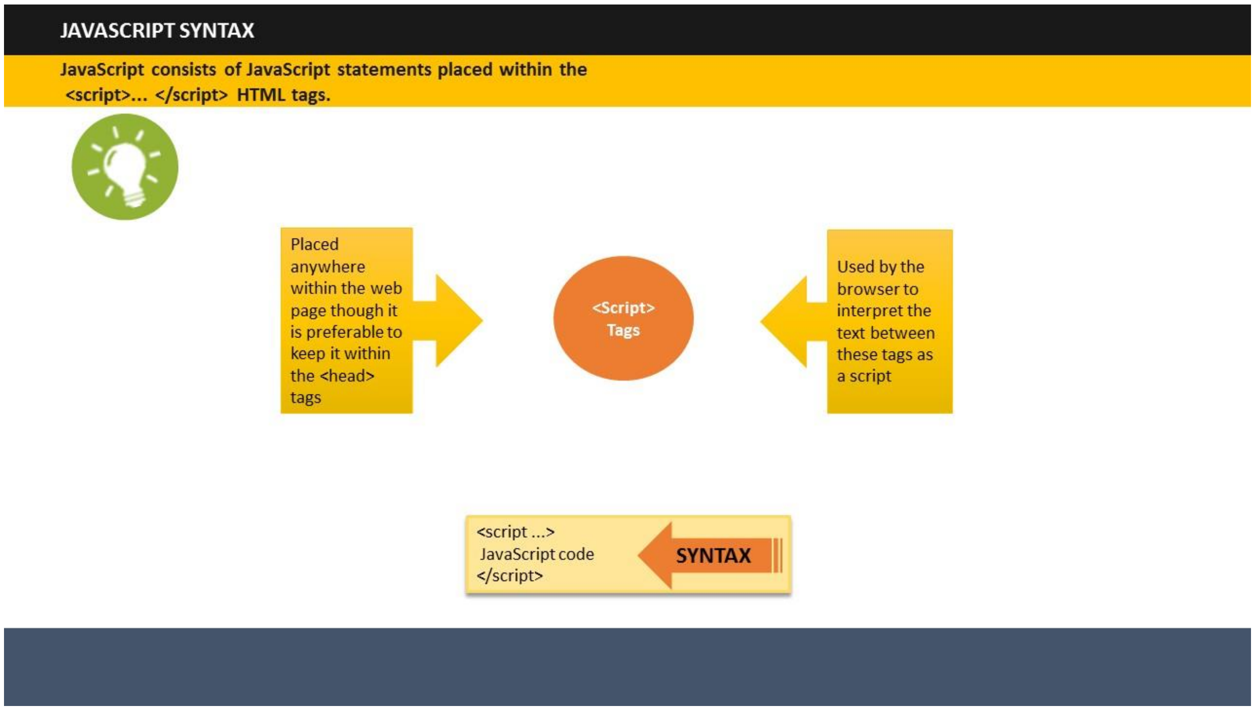
Ramkumar

**WHAT IS JAVASCRIPT?**

**JavaScript is a programming language that helps in adding interactivity to documents/webpages.**

JavaScript and Java are completely different languages, both in concept and design.

1. JavaScript is the most popular client side programming language in the world.



**JAVASCRIPT SYNTAX**

**JavaScript consists of JavaScript statements placed within the <script>... </script> HTML tags.**

Placed anywhere within the web page though it is preferable to keep it within the <head> tags

<Script> Tags

Used by the browser to interpret the text between these tags as a script

```
<script ...>
 JavaScript code
</script>
```

SYNTAX

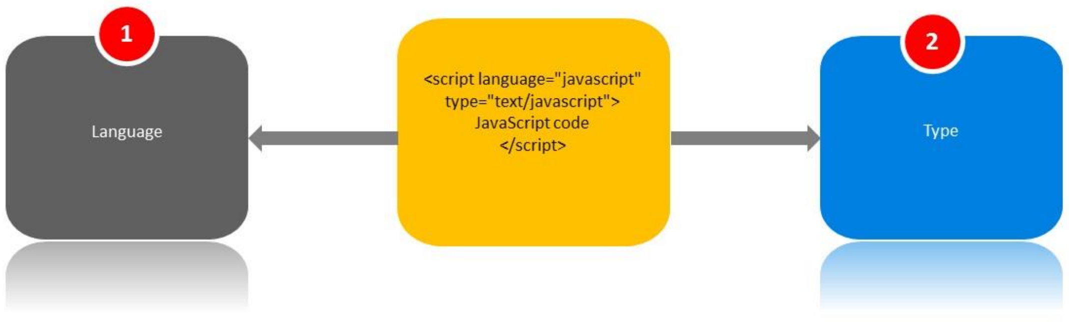
1. JavaScript consists of JavaScript statements placed within the <script>... </script> HTML tags.

2. The <script> tags are placed anywhere within the web page though it is preferable to keep it within the <head> tags. The <script> tags tell the browser to begin interpreting the text between these tags as a script.

3. The syntax of JavaScript looks like this:

```
<script ...>
 JavaScript code
</script>
```

## ATTRIBUTES OF SCRIPT TAGS


1
Language
<script language="javascript" type="text/javascript"> JavaScript code </script>
2
Type

1. A script tag has two attributes: Language and Type.

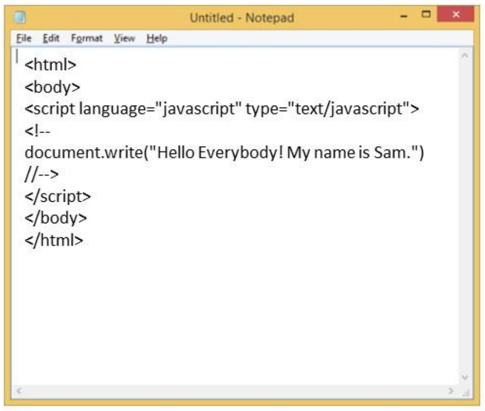
   a. ***Language****:* It specifies the scripting language. Generally, its value is “javascript”.

   b. ***Type****:* It specifies the scripting language in use. Its value is set to "text/javascript".

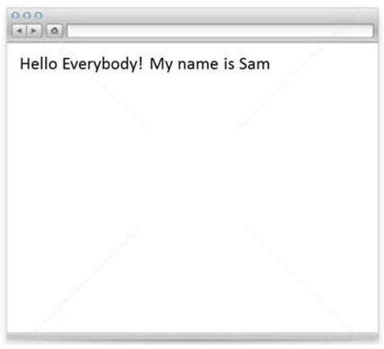
2. Using these two attributes the JavaScript segment will appear like this:

```
<script language="javascript" type="text/javascript">
 JavaScript code
</script>
```

## ATTRIBUTES OF SCRIPT TAGS



```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
document.write("Hello Everybody! My name is Sam.")
//-->
</script>
</body>
</html>
```



```
Hello Everybody! My name is Sam
```

```
<html>
<body>
<script language="javascript" type="text/javascript">
document.write("Hello Everybody! My name is Sam.")

</script>
</body>
</html>
```

## WHAT IS PROGRAMING ALL ABOUT?

**Programming is a way of making computers do what you want them to do.**

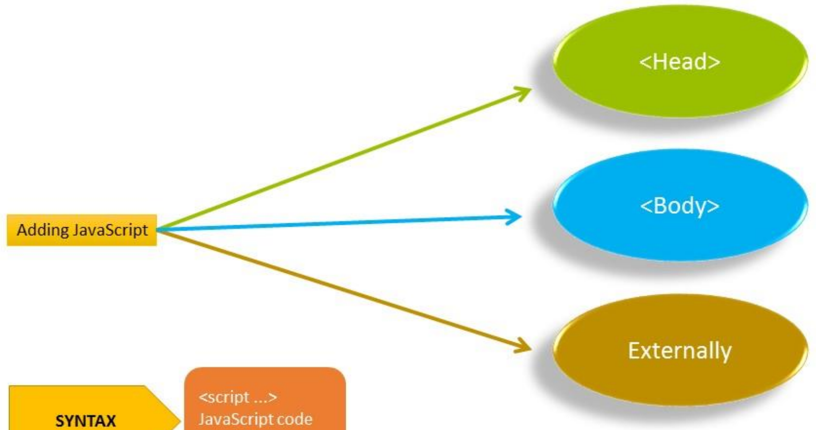
JavaScript is called as an Object Oriented Programming (OOP) language.

JavaScript is an extremely popular language amongst programmers and web designers.

1. JavaScript is also called as an Object Oriented Programming or OOP language. 2.
2. This is because JavaScript involves working with objects.
3. JavaScript can change HTML content, its attributes and style.
4. It can also be used to validate data inputs

## HOW TO ADD A SCRIPT IN YOUR PAGE?

**WHERE TO ADD THE JAVASCRIPT CODES?**

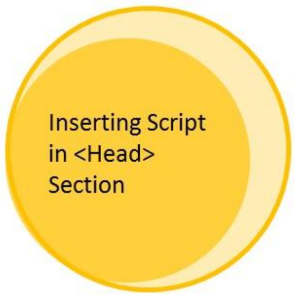


**SYNTAX**

```
<script ...>
JavaScript code
</script>
```

**HOW TO ADD A SCRIPT IN YOUR PAGE?**

**EXAMPLE**



```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "I am fine. Thank you!";
}
</script>
</head>
<body>
<h1>JavaScript in Head Section</h1>
<p id="demo">Hello! How are you?</p>
<button type="button" onclick="myFunction()">Try it</button>
</body>
</html>
```

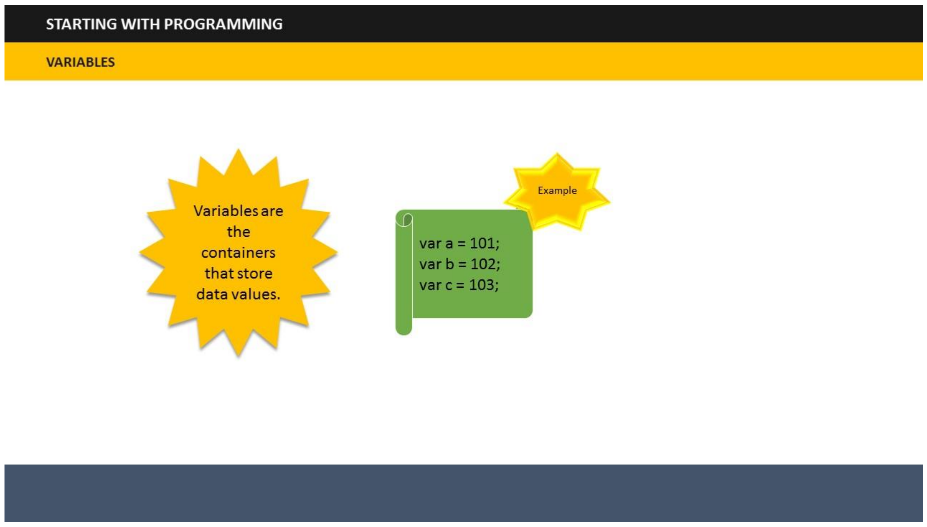
1. If you click the Try It button, the function is invoked in this script and it will give a result.
2. Result shown in the example: “I am fine. Thank you!”

**HOW TO ADD A SCRIPT IN YOUR PAGE?**

**EXAMPLE**

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript in Body Section</h1>

<p id="demo">Would you like to have tea or coffee?</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Coffee, please.";
}
</script>

</body>
</html>
```

➢ Result  shown in the example: “Coffee, please”



STARTING WITH PROGRAMMING

VARIABLES

Variables are the containers that store data values.
Example

```
var a = 101;
var b = 102;
var c = 103;
```

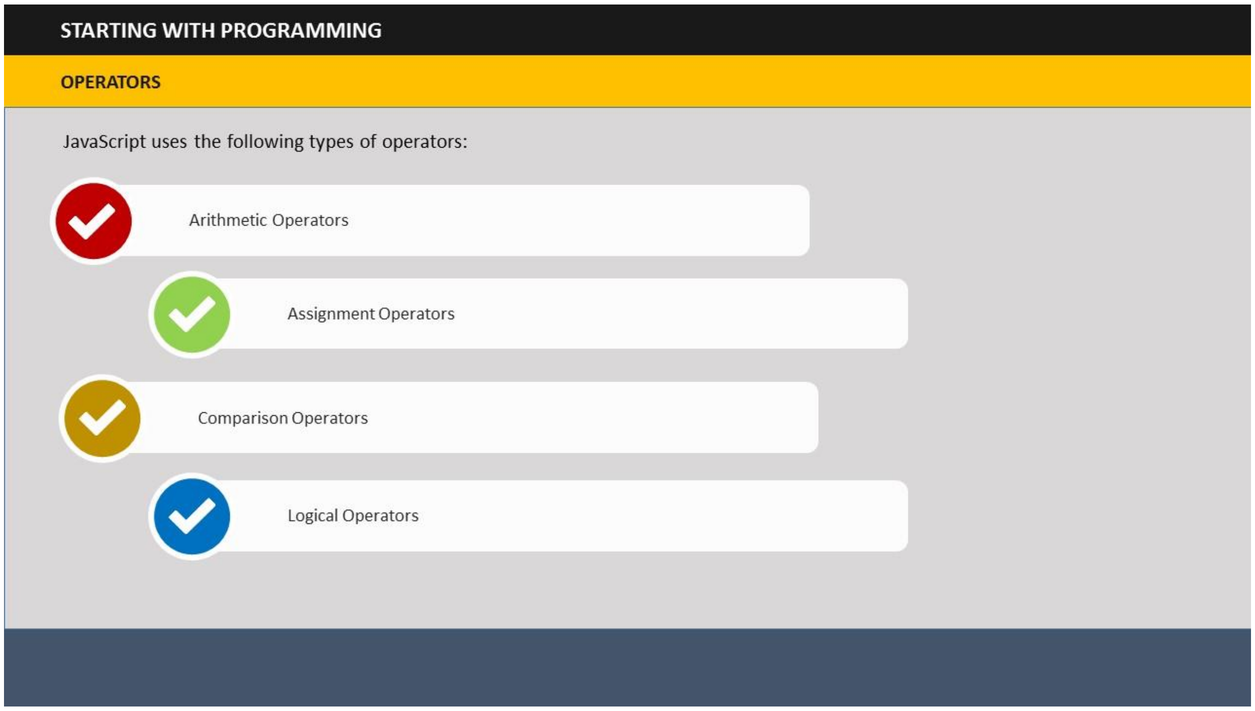
- JavaScript uses variables, operators and data types.
- Variables can be seen as containers that can store data values.
- You can place data into these containers and name these containers.
- Whenever, you wish to refer to the stored data, you can access the required container.

1. examples of variables.

```
var a = 101;
var b = 102;
var c = 103;
```

2. Here a, b, and c are the containers which are storing their corresponding values 101, 102, 103.

## OPERATORS



STARTING WITH PROGRAMMING

OPERATORS

JavaScript uses the following types of operators:

✓ Arithmetic Operators

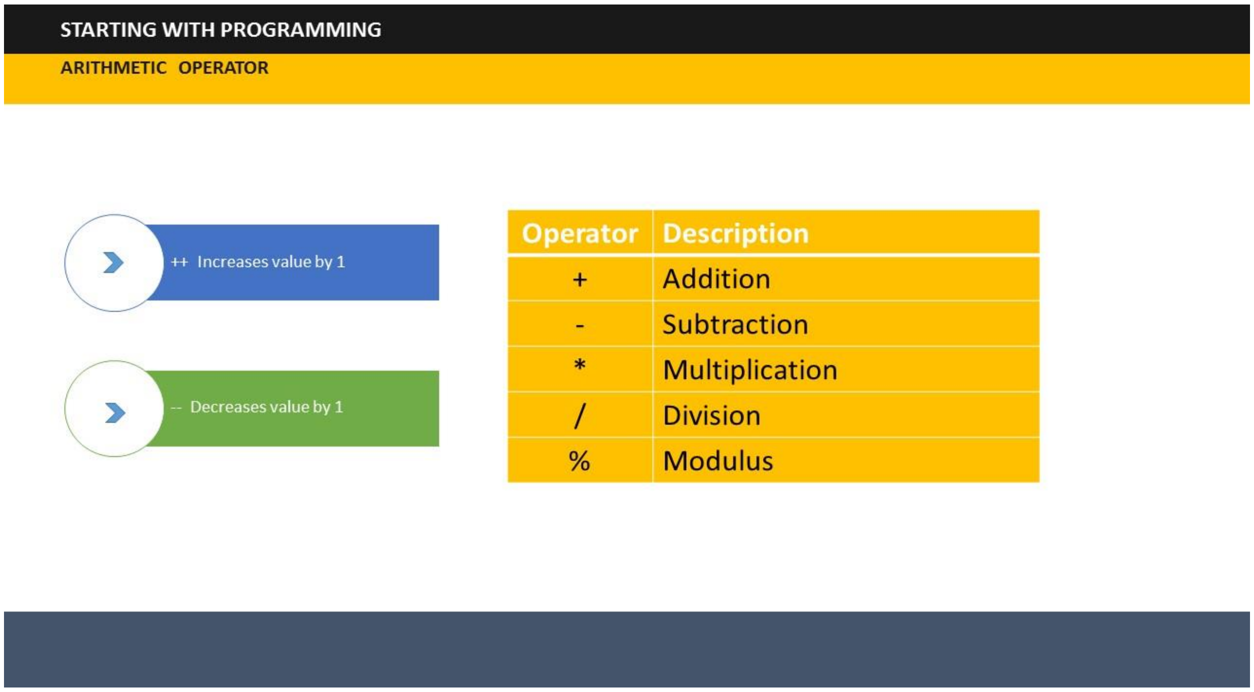
✓ Assignment Operators

✓ Comparison Operators

✓ Logical Operators

1. JavaScript uses the following types of operators:

   - ✓ Arithmetic Operators
   - ✓ Assignment Operators
   - ✓ Comparison Operators
   - ✓ Logical Operators



STARTING WITH PROGRAMMING

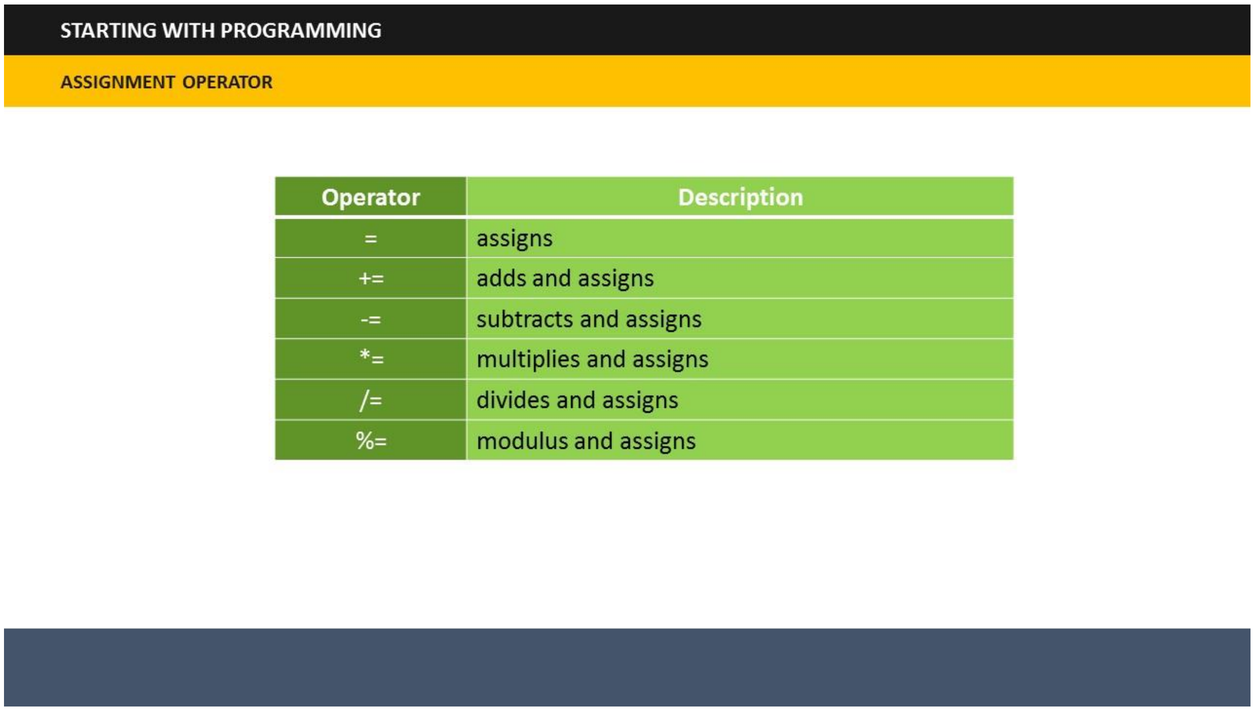
ARITHMETIC OPERATOR

++ Increases value by 1

-- Decreases value by 1

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |

- The arithmetic operators function as the addition, subtraction, multiplication, division, and modulus operators, and operate very similarly to other languages.

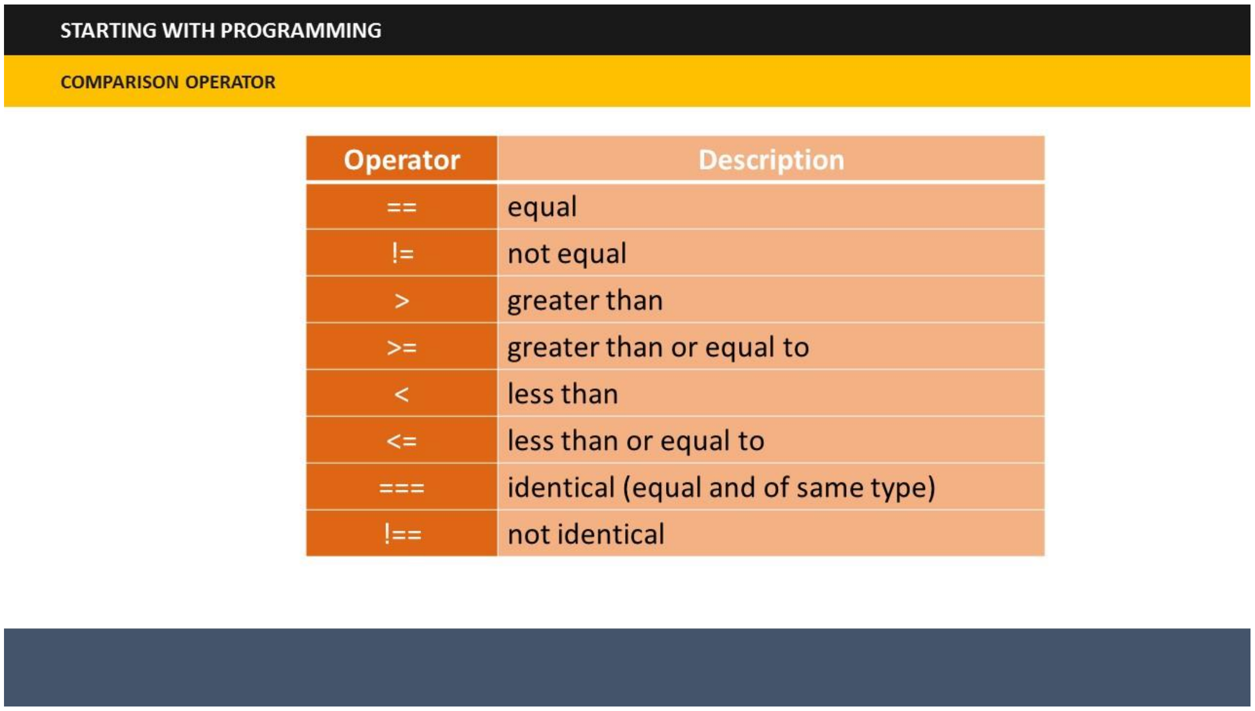


STARTING WITH PROGRAMMING

ASSIGNMENT OPERATOR

| Operator | Description |
|---|---|
| = | assigns |
| += | adds and assigns |
| -= | subtracts and assigns |
| *= | multiplies and assigns |
| /= | divides and assigns |
| %= | modulus and assigns |

The assignment operators assign a value to a variable.

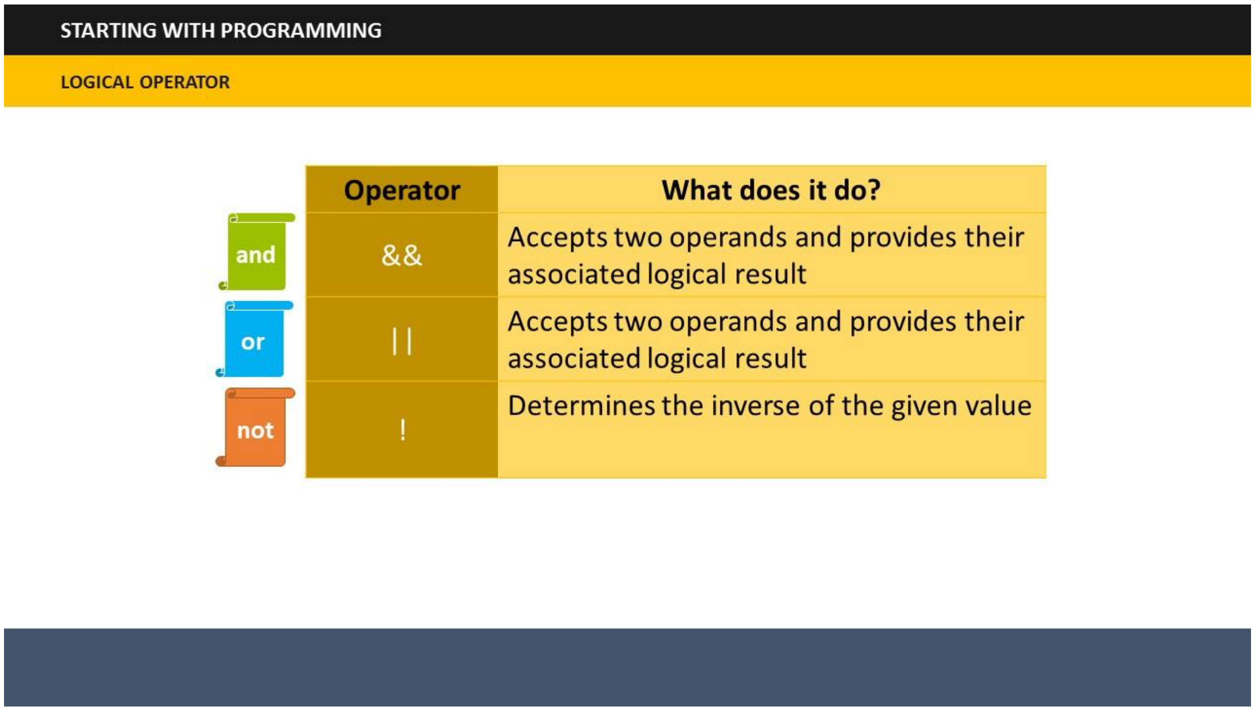


STARTING WITH PROGRAMMING

COMPARISON OPERATOR

| Operator | Description |
|---|---|
| == | equal |
| != | not equal |
| > | greater than |
| >= | greater than or equal to |
| < | less than |
| <= | less than or equal to |
| === | identical (equal and of same type) |
| !== | not identical |

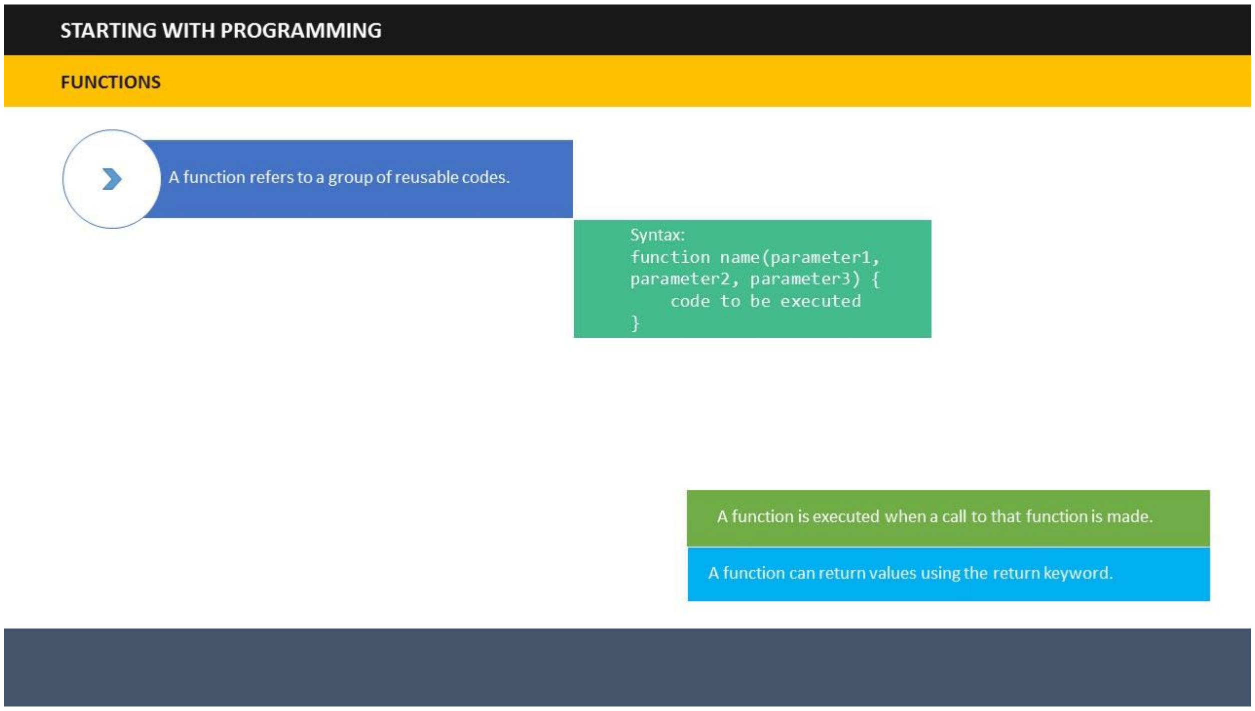
## FUNCTIONS



STARTING WITH PROGRAMMING

LOGICAL OPERATOR

| | Operator | What does it do? |
|---|---|---|
| and | && | Accepts two operands and provides their associated logical result |
| or | \|\| | Accepts two operands and provides their associated logical result |
| not | ! | Determines the inverse of the given value |



STARTING WITH PROGRAMMING

FUNCTIONS

A function refers to a group of reusable codes.

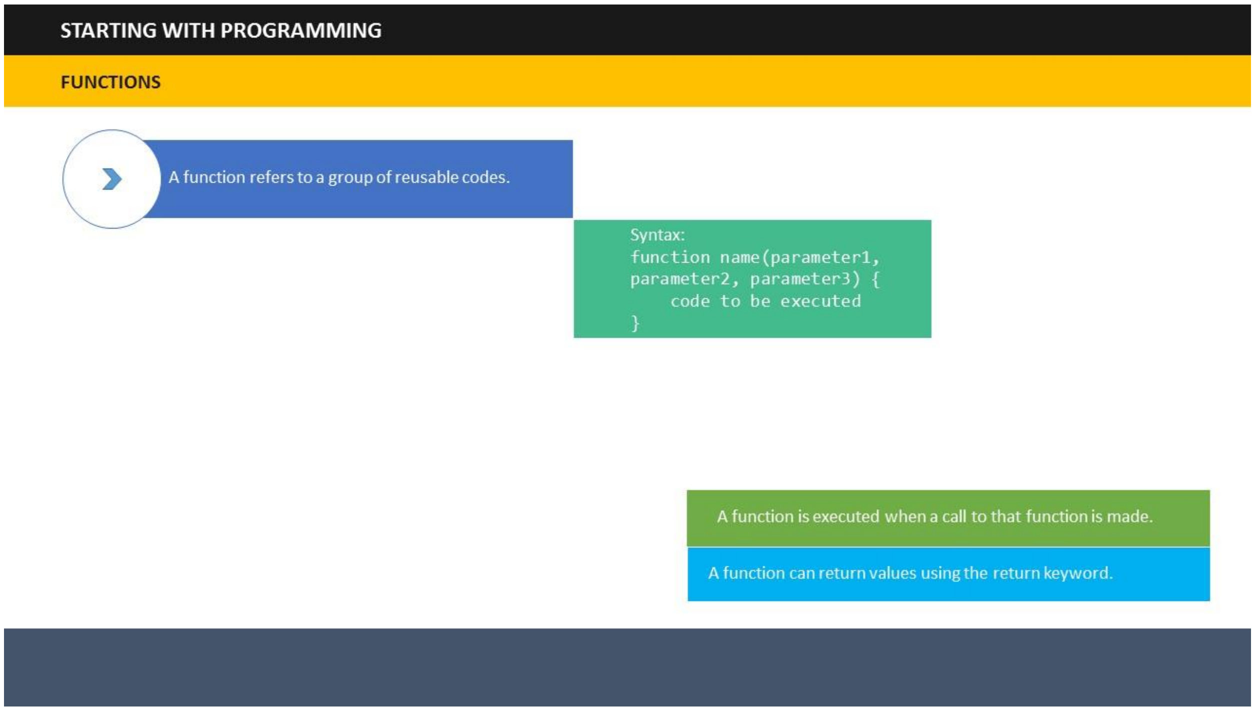
```
Syntax:
function name(parameter1,
parameter2, parameter3) {
    code to be executed
}
```

A function is executed when a call to that function is made.

A function can return values using the return keyword.

1. A Function in JavaScript refers to a group of reusable codes.
2. This helps in saving time as a web designer will not be required to write the same codes over and over again.

3. You can write a function using the syntax
4. A function is executed when a call to that function is made anywhere within the script, the page, an external page, or by an event.
5. Another point to note is that functions can return values using the return keyword.



STARTING WITH PROGRAMMING

FUNCTIONS

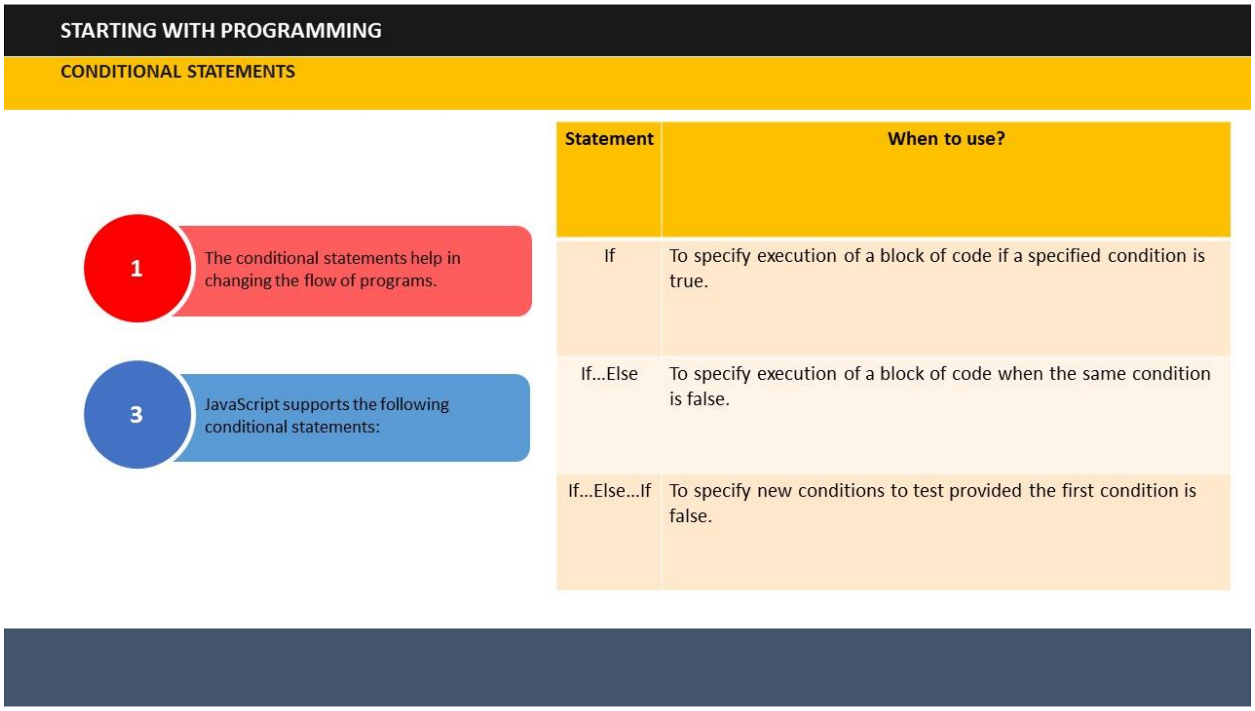
A function refers to a group of reusable codes.

Syntax:
```
function name(parameter1,
parameter2, parameter3) {
    code to be executed
}
```

A function is executed when a call to that function is made.

A function can return values using the return keyword.

## CONDITIONAL STATEMENTS



STARTING WITH PROGRAMMING

CONDITIONAL STATEMENTS

1 The conditional statements help in changing the flow of programs.

3 JavaScript supports the following conditional statements:

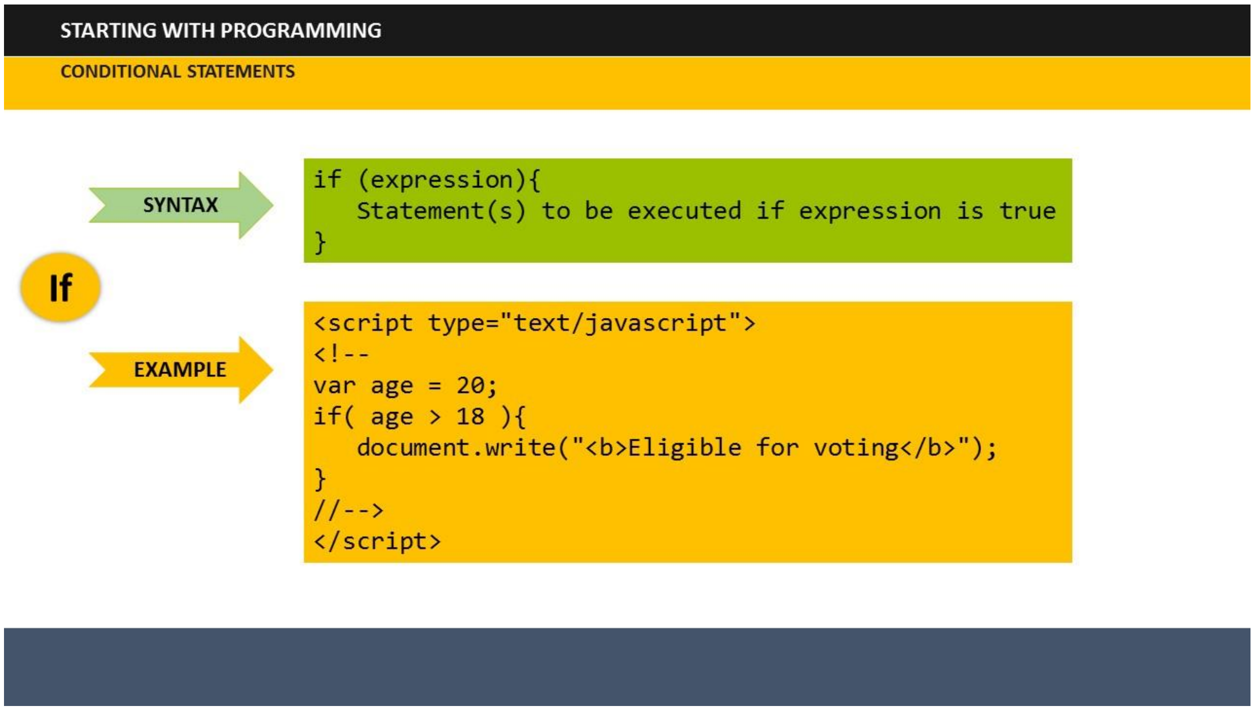
| Statement | When to use? |
|---|---|
| If | To specify execution of a block of code if a specified condition is true. |
| If…Else | To specify execution of a block of code when the same condition is false. |
| If…Else…If | To specify new conditions to test provided the first condition is false. |

1. The JavaScript conditional statements help in changing the flow of programs. In other words, they allow different actions to be taken for different decisions.
2. JavaScript supports the following conditional statements:

   - ✓ **If**: You can use this statement to specify execution of a block of code if a specified condition is true.
   - ✓ **If…Else**: You can use this statement to specify execution of a block of code when the same condition is false.
   - ✓ **If…Else…If**: You can use this statement to specify new conditions to test provided the first condition is false.

3. Let’s look at the examples showing how to use these statements.

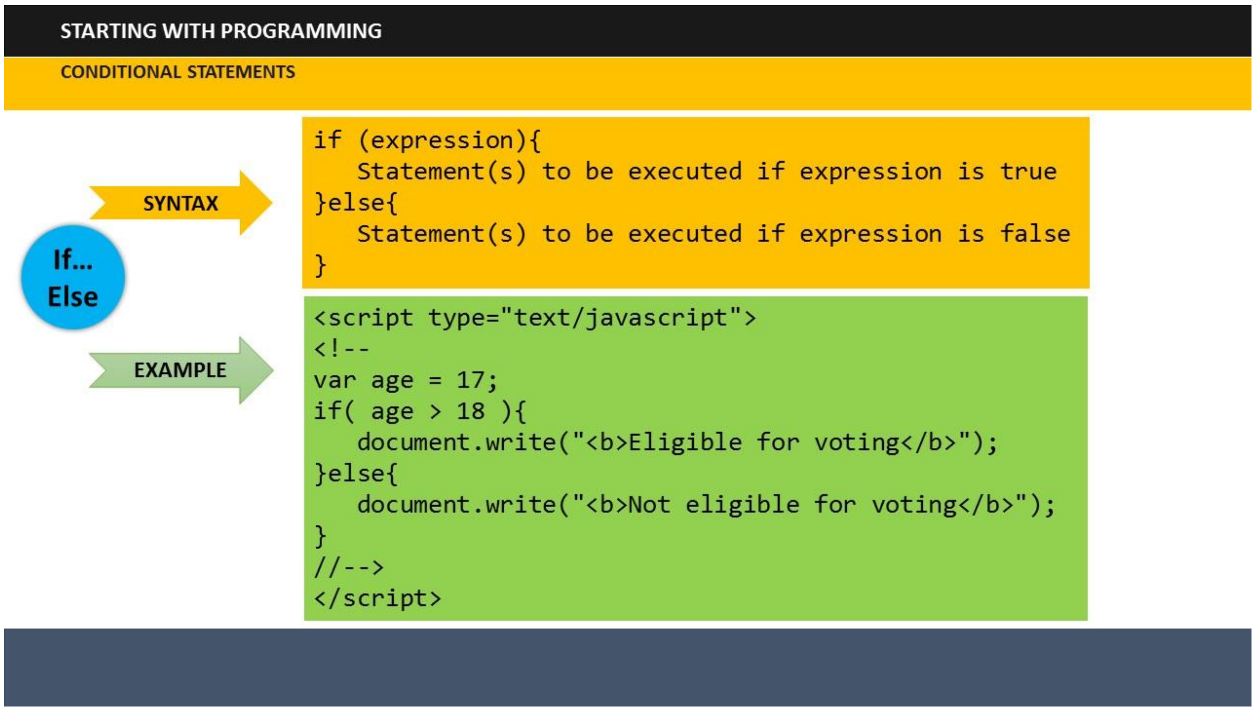
4. The syntax for if statement is shown on the screen. The example shows how if statements can be used to decide if a person is eligible for voting or not basis the age. If the entered age is greater than 18, the output will say “Eligible for voting”. In this case, the variable is 20 which obviously is more than 18, so the result would say, Eligible for voting.



```
if (expression){
   Statement(s) to be executed if expression is true
}
```

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
   document.write("<b>Eligible for voting</b>");
}
//-->
</script>
```

1. Next, we have the syntax for if…then statement. In the previous example, the variable was 20, so the result was true. But, if the variable is 17 which is less than 18, the statement will show the output associated with else part of the statement. That is “Not eligible for voting”.

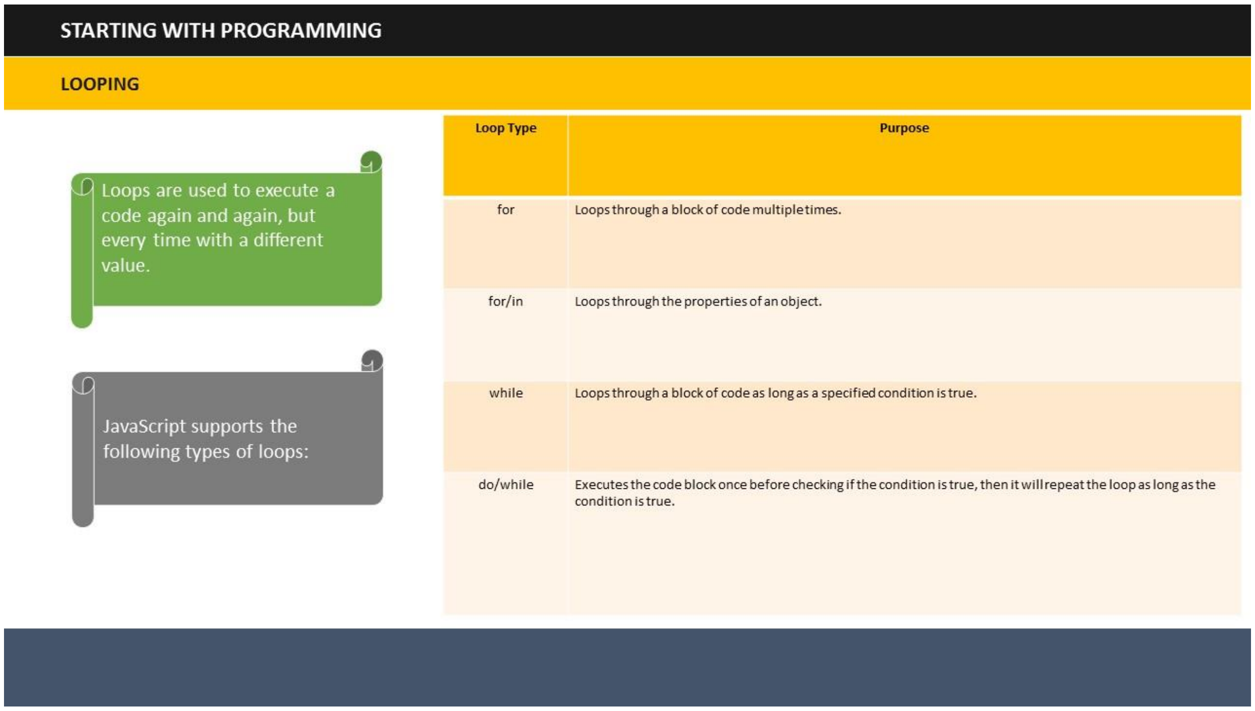

STARTING WITH PROGRAMMING
CONDITIONAL STATEMENTS
If…
Else
SYNTAX
EXAMPLE

```
if (expression){
   Statement(s) to be executed if expression is true
}else{
   Statement(s) to be executed if expression is false
}
```

```
<script type="text/javascript">
<!--
var age = 17;
if( age > 18 ){
   document.write("<b>Eligible for voting</b>");
}else{
   document.write("<b>Not eligible for voting</b>");
}
//-->
</script>
```

Lastly, take a look at the syntax of If…Else…If statement. Here the entered variable is Basketball. The if statement will be false for this case since it defines tennis ball, while the variable is cricket ball. So, the decision goes to the first else…if statement which is the defining condition for cricket ball. Since the first else…if statement also doesn't match, so the decision is passed to the second statement. The second else…if statement is defining the basketball, so our variable input matches with it. Hence the result on the screen would show as Basketball.

# LOOPING



STARTING WITH PROGRAMMING

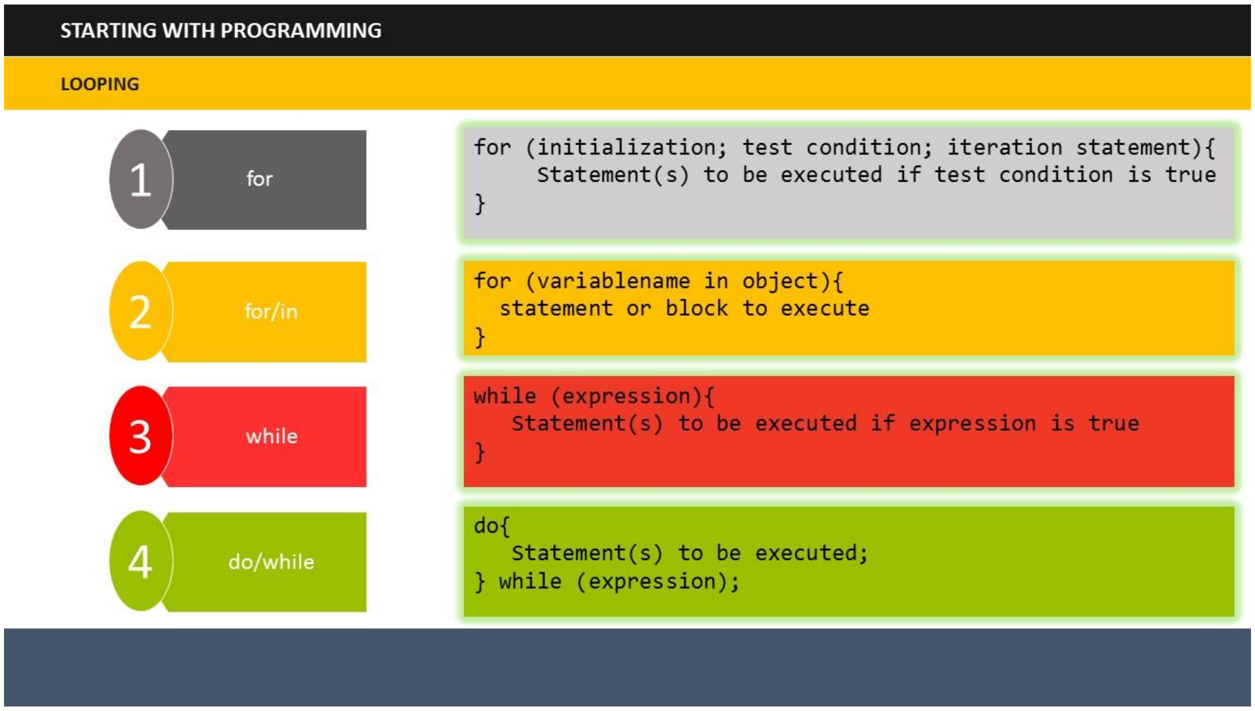
LOOPING

Loops are used to execute a code again and again, but every time with a different value.

JavaScript supports the following types of loops:

| Loop Type | Purpose |
|---|---|
| for | Loops through a block of code multiple times. |
| for/in | Loops through the properties of an object. |
| while | Loops through a block of code as long as a specified condition is true. |
| do/while | Executes the code block once before checking if the condition is true, then it will repeat the loop as long as the condition is true. |

1. Loops are used to execute a code again and again, but every time with a different value.
2. JavaScript supports the following types of loops:
   - ✓ For
   - ✓ For/in
   - ✓ While
   - ✓ Do/while
3. The for loop loops through a block of code multiple times.
4. The for/in loop loops through the properties of an object.
5. The while loop loops through a block of code as long as a specified condition is true.
6. The do/while loop executes the code block once before checking if the condition is true, then it will repeat the loop as long as the condition is true.



STARTING WITH PROGRAMMING

LOOPING

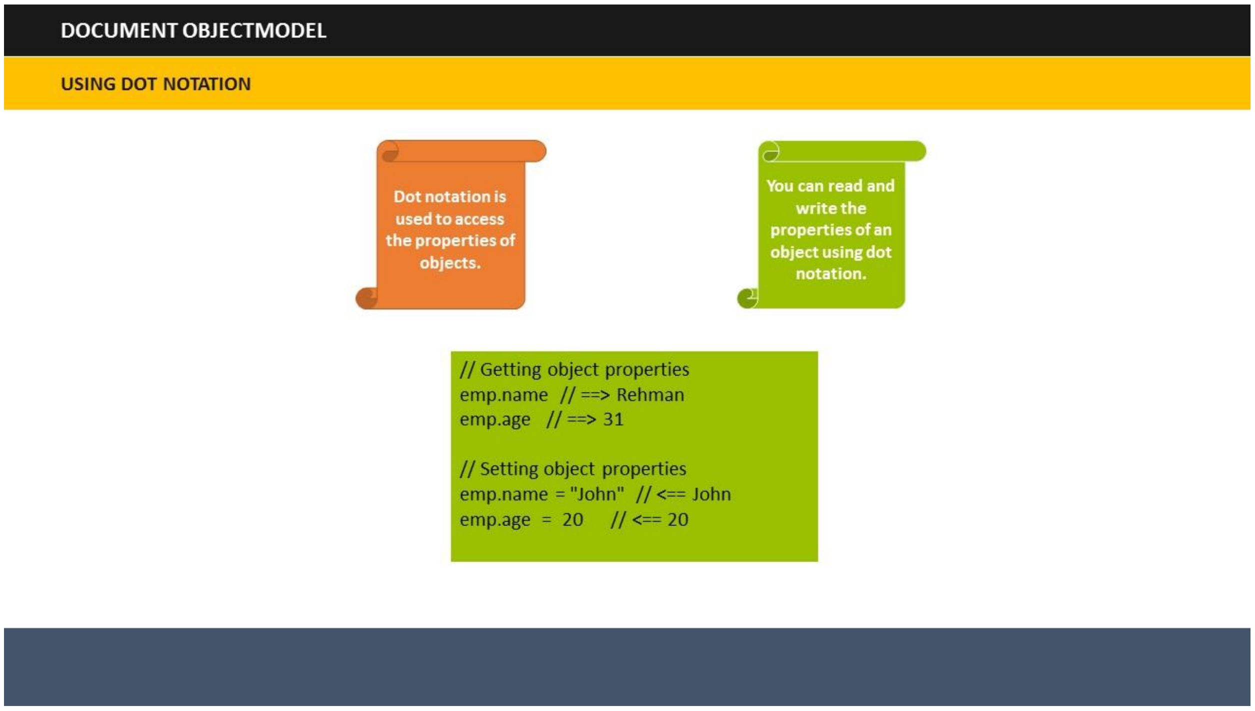
```
for (initialization; test condition; iteration statement){
     Statement(s) to be executed if test condition is true
}
```

```
for (variablename in object){
  statement or block to execute
}
```

```
while (expression){
   Statement(s) to be executed if expression is true
}
```

```
do{
   Statement(s) to be executed;
} while (expression);
```

1. Take a look at the syntax given on the screen for each of these loops.



DOCUMENT OBJECTMODEL

USING DOT NOTATION

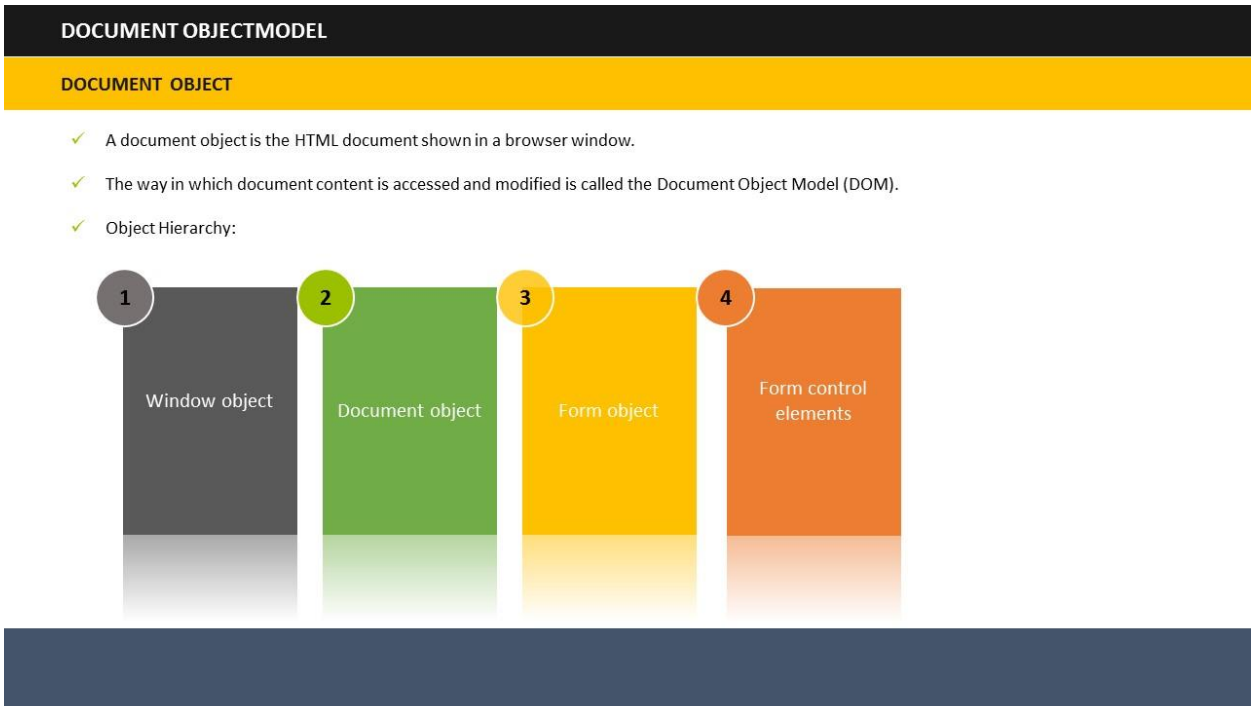
Dot notation is used to access the properties of objects.

You can read and write the properties of an object using dot notation.

```
// Getting object properties
emp.name  // ==> Rehman
emp.age   // ==> 31

// Setting object properties
emp.name = "John"  // <== John
emp.age  =  20     // <== 20
```

1. Dot notation is used to access the properties of objects. You can read and write the properties of an object using dot notation.
2. Take a look at the example given on the screen.

## SUBTOPIC 2: DOCUMENT OBJECT


DOCUMENT OBJECTMODEL

DOCUMENT OBJECT

- ✓ A document object is the HTML document shown in a browser window.
- ✓ The way in which document content is accessed and modified is called the Document Object Model (DOM).
- ✓ Object Hierarchy:

1
Window object
2
Document object
3
Form object
4
Form control elements

1. A document object is the HTML document shown in a browser window.
2. The way in which document content is accessed and modified is called the Document Object Model or DOM.
3. The objects are generally organized in a hierarchy in an HTML document.
4. Let’s take a look the hierarchy of these objects.
5. First is the window object which is at the top level in the hierarchy.
6. Next is the document object. It contains the contents of a page. Each HTML document loaded into a browser window becomes a document object.
7. Then comes the form object. Any object that is enclosed in the <form> tags falls into the category of form objects.
8. The last are the form control elements. They contain all the elements that define a form object. For example, radio buttons, text fields, checkboxes etc.

## DOM (Document Object Model)

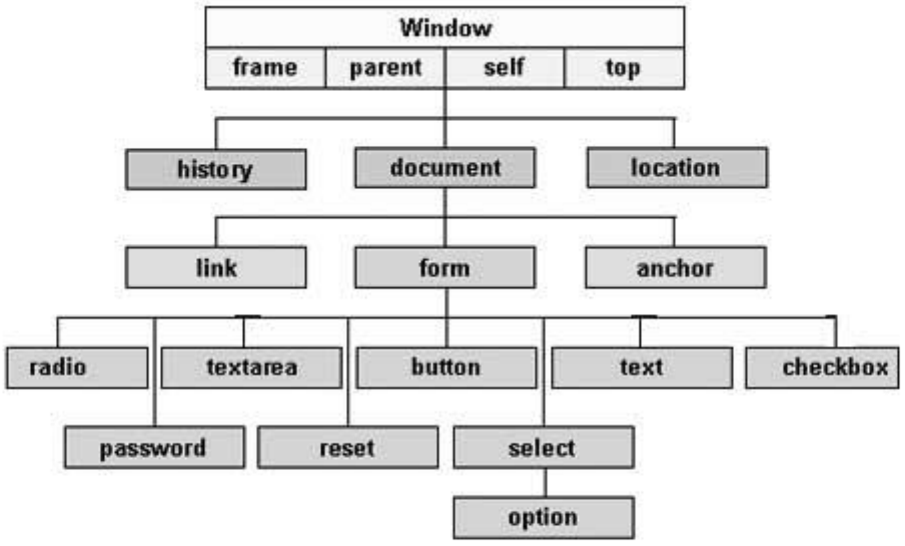
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

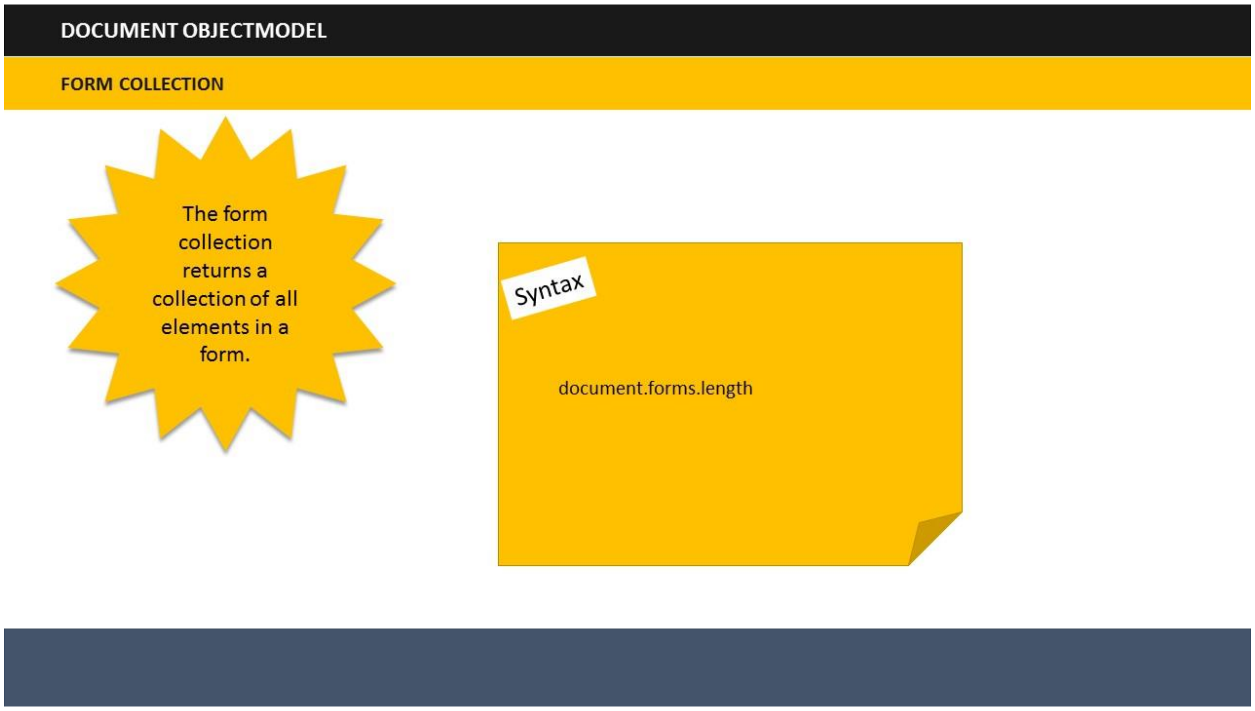
The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** – Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** – Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** – Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** – The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects –



## FORM COLLECTION



DOCUMENT OBJECTMODEL

FORM COLLECTION

The form collection returns a collection of all elements in a form.

Syntax

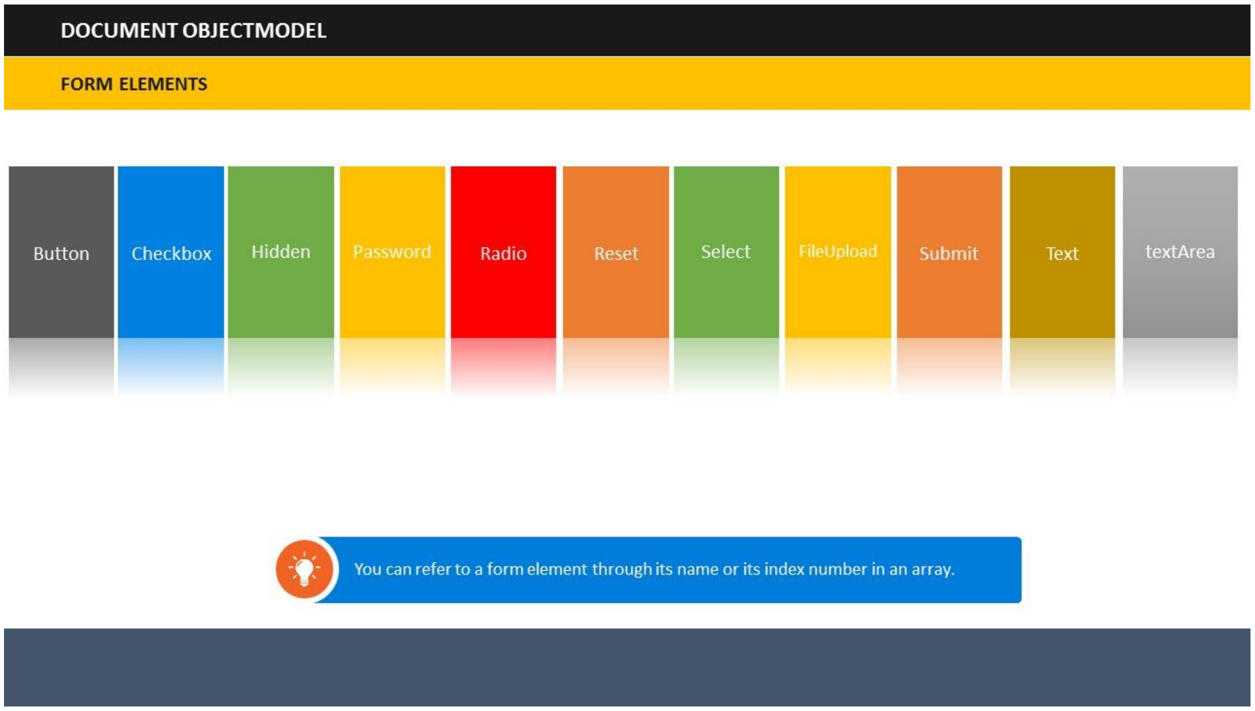
```
document.forms.length
```

1. Document objects have certain properties.

2. One of such properties is form elements collection. The form collection returns a collection of all elements in a form.

3. The syntax for forms collection command is document.forms.length

## FORM ELEMENTS



**TYPES OF FORM ELEMENTS**

### Form elements can be of several types such as:

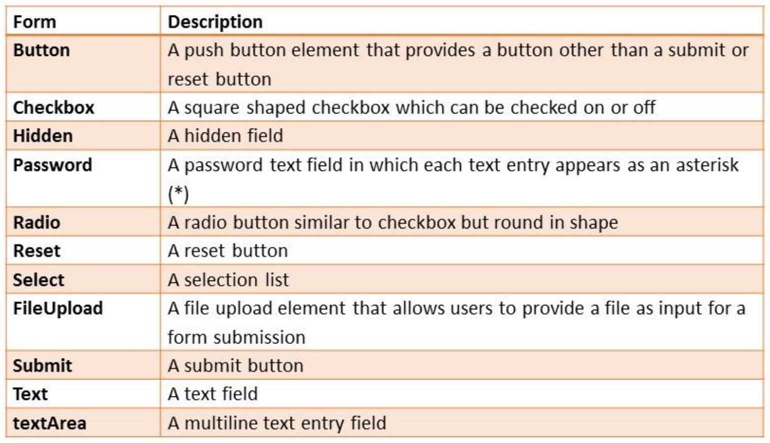
- ✓ Button
- ✓ Checkbox
- ✓ Hidden
- ✓ Password
- ✓ Radio
- ✓ Reset
- ✓ Select
- ✓ Submit
- ✓ Text
- ✓ textArea

You can refer to a form element through its name or its index number in an array.

Each of these elements has its own properties in JavaScript.

## DOCUMENT OBJECTMODEL

### FORM ELEMENTS

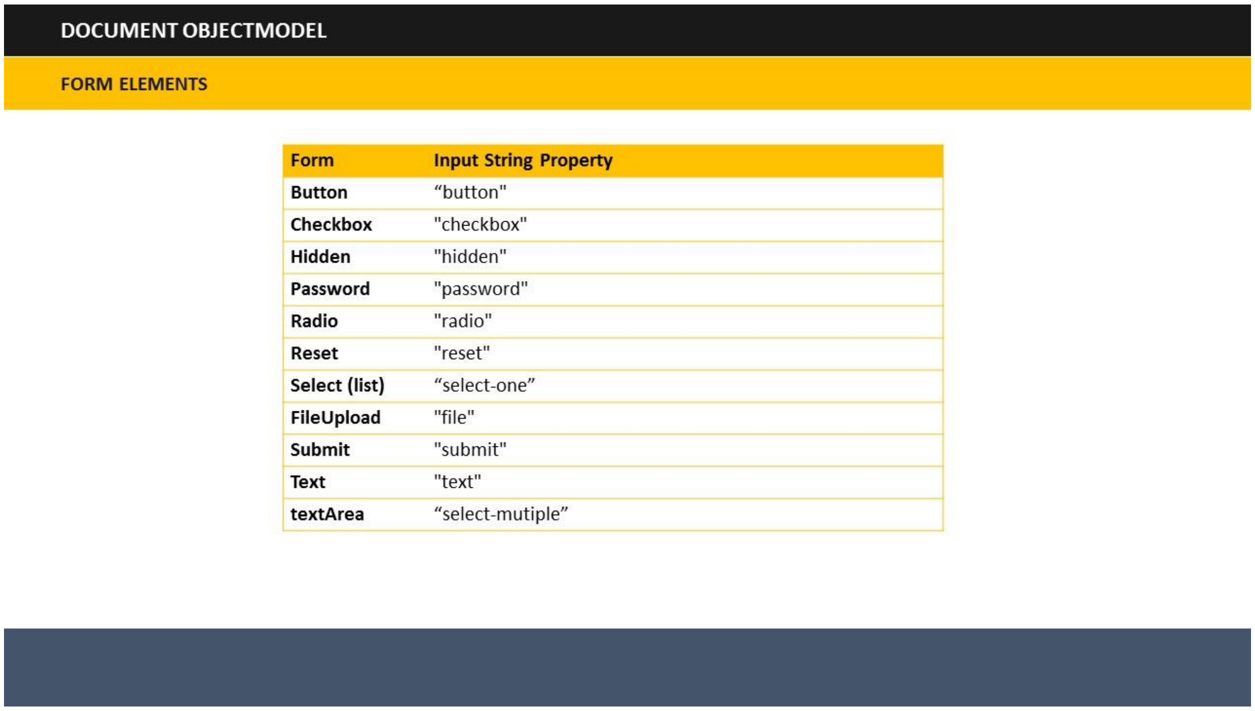


| Form | Description |
|---|---|
| **Button** | A push button element that provides a button other than a submit or reset button |
| **Checkbox** | A square shaped checkbox which can be checked on or off |
| **Hidden** | A hidden field |
| **Password** | A password text field in which each text entry appears as an asterisk (*) |
| **Radio** | A radio button similar to checkbox but round in shape |
| **Reset** | A reset button |
| **Select** | A selection list |
| **FileUpload** | A file upload element that allows users to provide a file as input for a form submission |
| **Submit** | A submit button |
| **Text** | A text field |
| **textArea** | A multiline text entry field |

**DESCRIPTION OF FORM ELEMENTS**

1. Button -> This provides a push button that provides functionality other than submit or reset button.
2. Checkbox -> This provides a square shaped checkbox which can be checked on or off.
3. Hidden -> This provides a hidden field.
4. Password -> This provides a password text field in which each text entry appears as an asterisk.
5. Radio -> This provides a radio button similar to checkbox but round in shape.
6. Reset -> This provides a reset button.
7. Select -> This provides a selection list.
8. FileUpload -> This provides a file upload element that allows users to provide a file as input for a form submission.
9. Submit -> This provides a submit button.
10. Text -> This provides a text field.
11. textArea -> This provides a multiline text entry field.

## IMAGES COLLECTION



DOCUMENT OBJECTMODEL

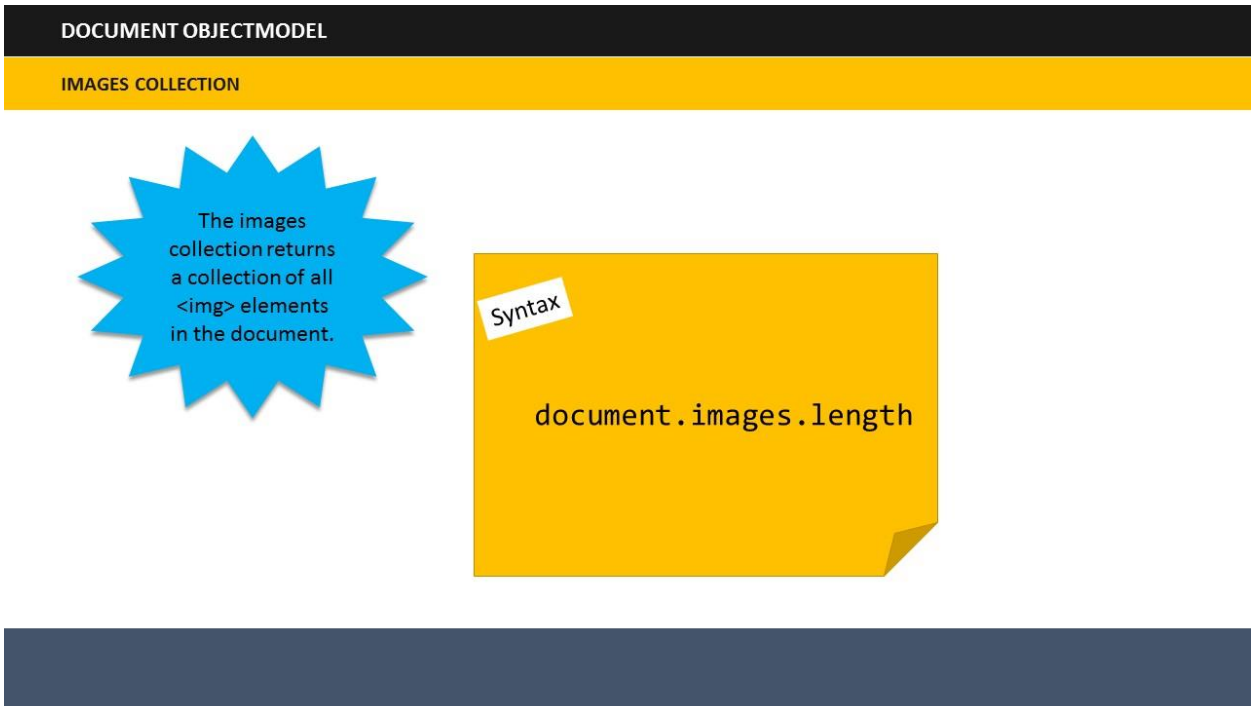
FORM ELEMENTS

| Form | Input String Property |
|---|---|
| Button | “button" |
| Checkbox | "checkbox" |
| Hidden | "hidden" |
| Password | "password" |
| Radio | "radio" |
| Reset | "reset" |
| Select (list) | “select-one” |
| FileUpload | "file" |
| Submit | "submit" |
| Text | "text" |
| textArea | “select-mutiple” |

**TYPE PROPERTIES**

1. Each of the form elements has a type property. This property is a string value that shows the type of input element.

2. The strings reflected by the various type properties for each form element are shown on the screen.



DOCUMENT OBJECTMODEL

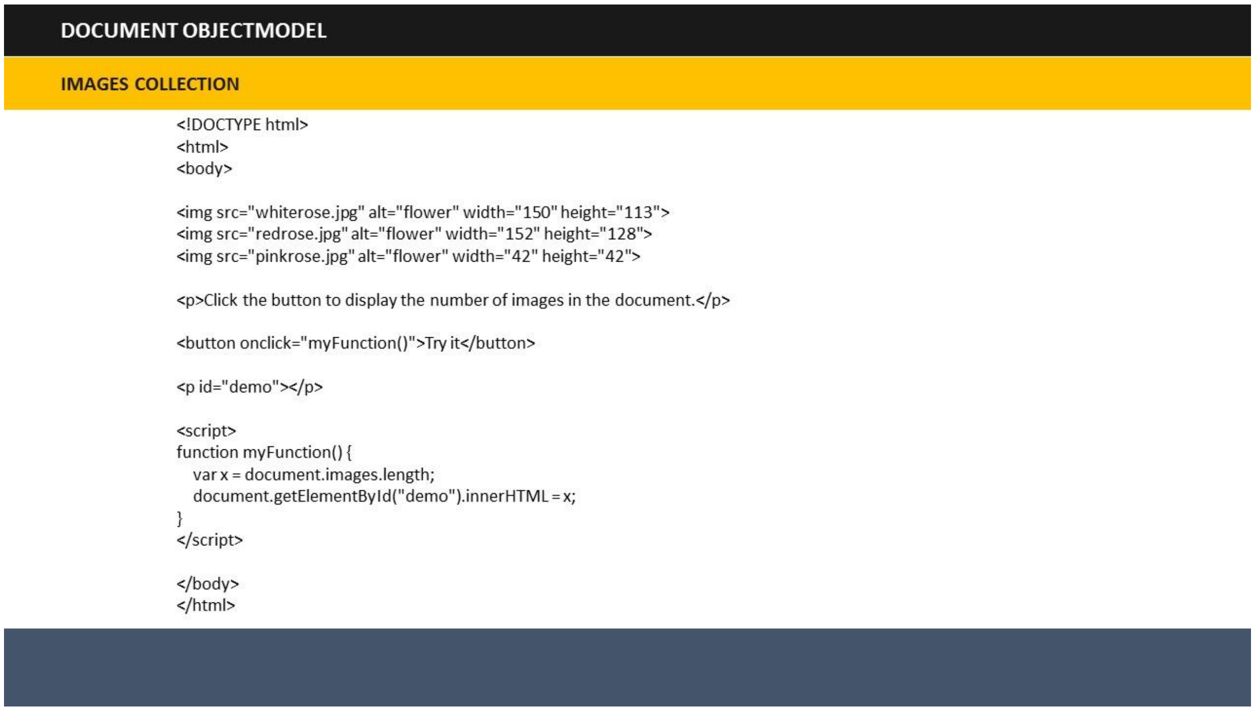
IMAGES COLLECTION

The images collection returns a collection of all <img> elements in the document.

Syntax

```
document.images.length
```

1. The images collection returns a collection of all <img> elements in the document.
2. The syntax for image collection is document.images.length

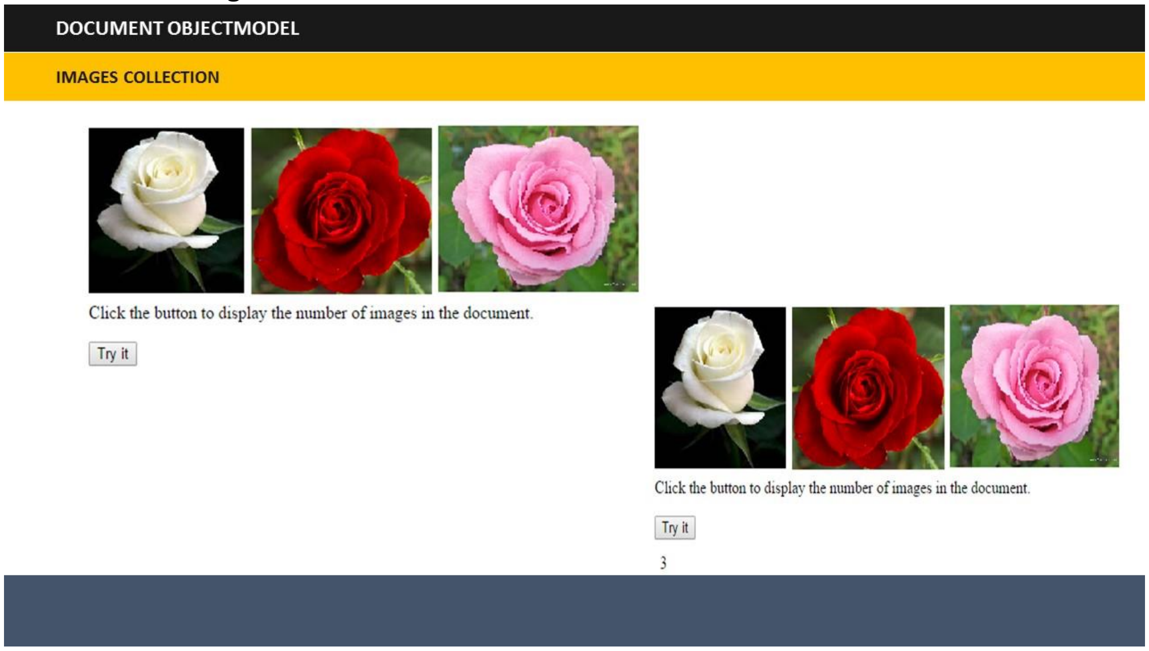


DOCUMENT OBJECTMODEL

IMAGES COLLECTION

```
<!DOCTYPE html>
<html>
<body>

<img src="whiterose.jpg" alt="flower" width="150" height="113">
<img src="redrose.jpg" alt="flower" width="152" height="128">
<img src="pinkrose.jpg" alt="flower" width="42" height="42">

<p>Click the button to display the number of images in the document.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
   var x = document.images.length;
   document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```

1. For example, take a look at the script shown on the screen. It shows that the page has three images.



DOCUMENT OBJECTMODEL

IMAGES COLLECTION

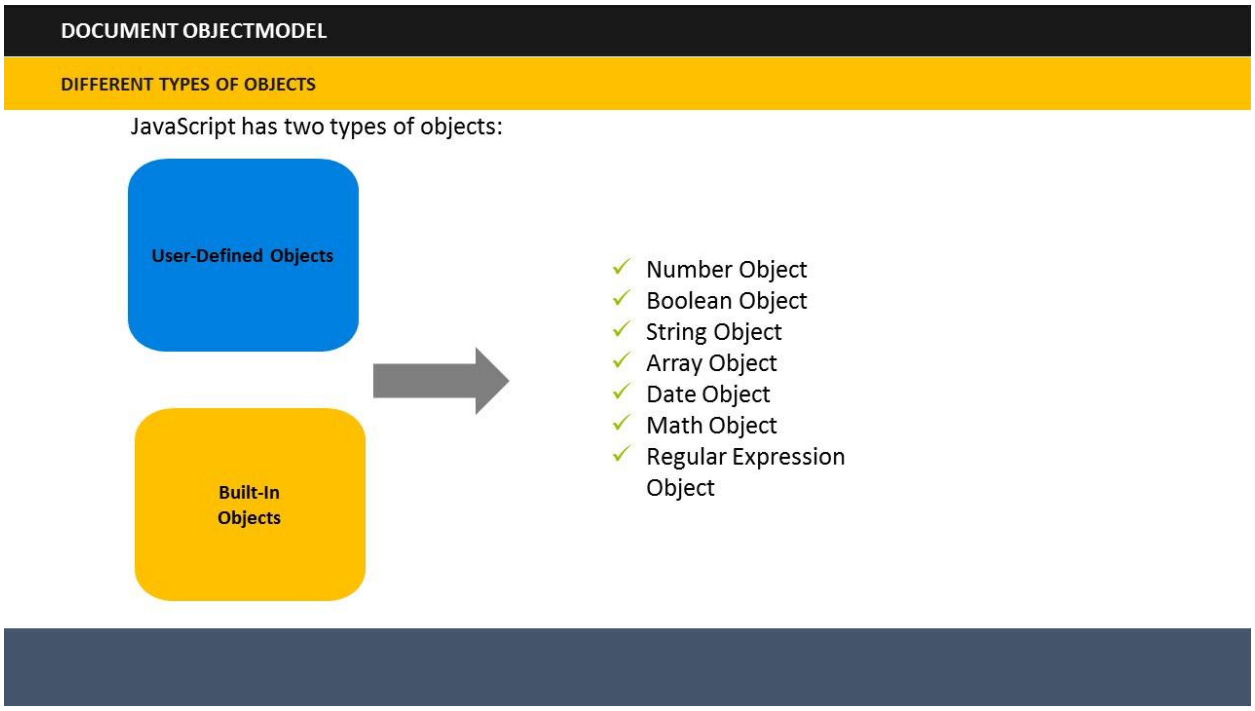
Click the button to display the number of images in the document.

Click the button to display the number of images in the document.

3

1. This script will display the result as shown on the screen and when you will click the Try It button, the image collection code will show you the result as 3 which is the actual number of images in the HTML page.

## SUBTOPIC 6: DIFFERENT TYPES OF OBJECTS



DOCUMENT OBJECTMODEL

DIFFERENT TYPES OF OBJECTS

JavaScript has two types of objects:

User-Defined Objects

Built-In Objects

- ✓ Number Object
- ✓ Boolean Object
- ✓ String Object
- ✓ Array Object
- ✓ Date Object
- ✓ Math Object
- ✓ Regular Expression Object

1. JavaScript has two types of objects: User-Defined Objects and Built-In Objects

2. The user-defined objects are created using a constructor function called Object(). The user-defined objects can be created using the Object() constructor. The constructor initializes a new object and assigns properties to it.

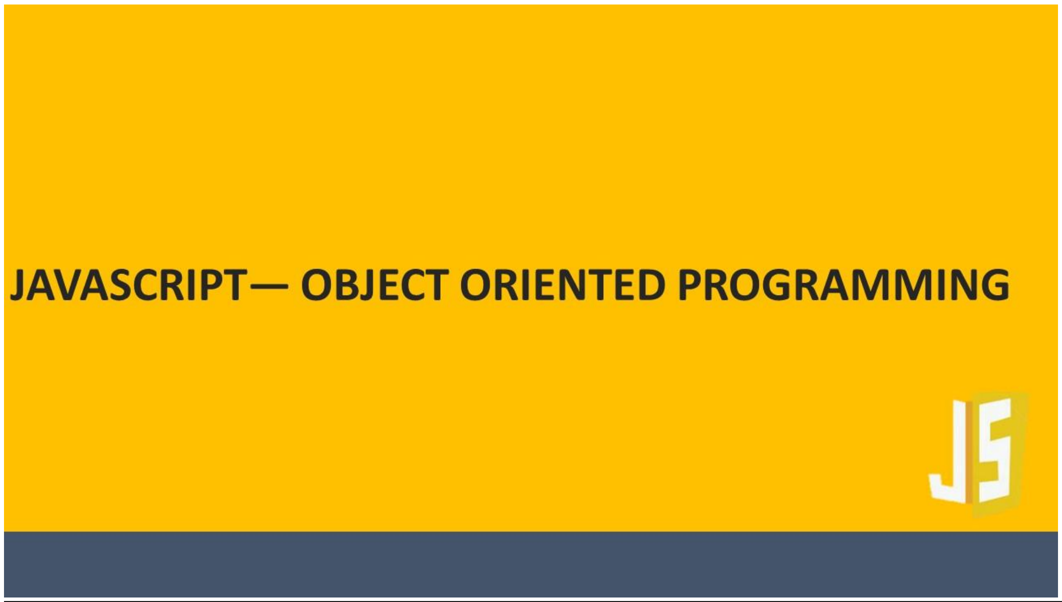
3. The built-in objects can be accessed from anywhere in your program. They will function in the same way irrespective of the browser used.

4. The available built-in objects are:

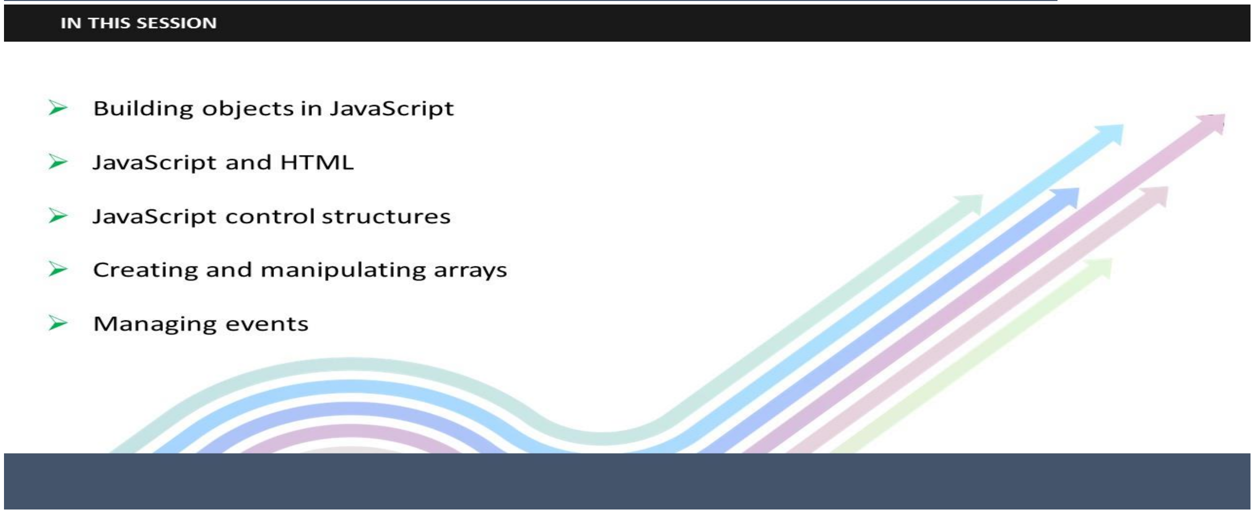
   - ✓ Number Object
   - ✓ Boolean Object
   - ✓ String Object
   - ✓ Array Object
   - ✓ Date Object
   - ✓ Math Object
   - ✓ Regular Expression Object
   - Regular Expression Object

# TOPIC 7 – OBJECT ORIENTED PROGRAMMING

## IN THIS SESSION – SESSION OBJECTIVES



JAVASCRIPT— OBJECT ORIENTED PROGRAMMING



IN THIS SESSION

- ➢ Building objects in JavaScript
- ➢ JavaScript and HTML
- ➢ JavaScript control structures
- ➢ Creating and manipulating arrays
- ➢ Managing events

1. building objects in JavaScript and…
2. Working of JavaScript and HTML.
3. Data Types, Variables, Operators and Control Structures in JavaScript.
4. In addition, you will also learn about using loops....
5. creating and manipulating arrays and…
6. using object hierarchy.
7. Finally, you will learn about managing events.

8.

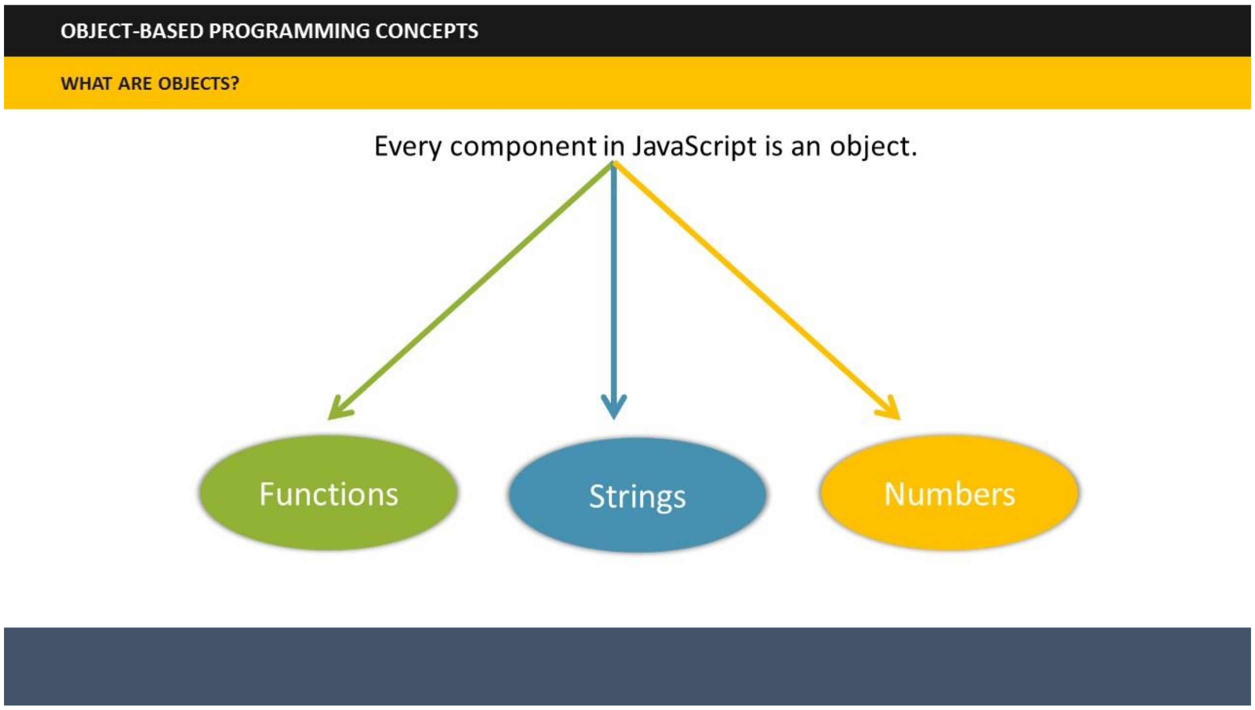
OBJECT-BASED PROGRAMMING CONCEPTS

WHAT ARE OBJECTS?

JavaScript is known as an Object Oriented Programming (OOP) language.

1. By now, you already have a basic understanding of JavaScript. In this session, we will build up on this understanding by going through a few key concepts in detail. One of these concepts is significance of objects in JavaScript.

2. JavaScript is called as an Object Oriented Programming or OOP language. In other words, we can say that using JavaScript involves working with what are known as objects.

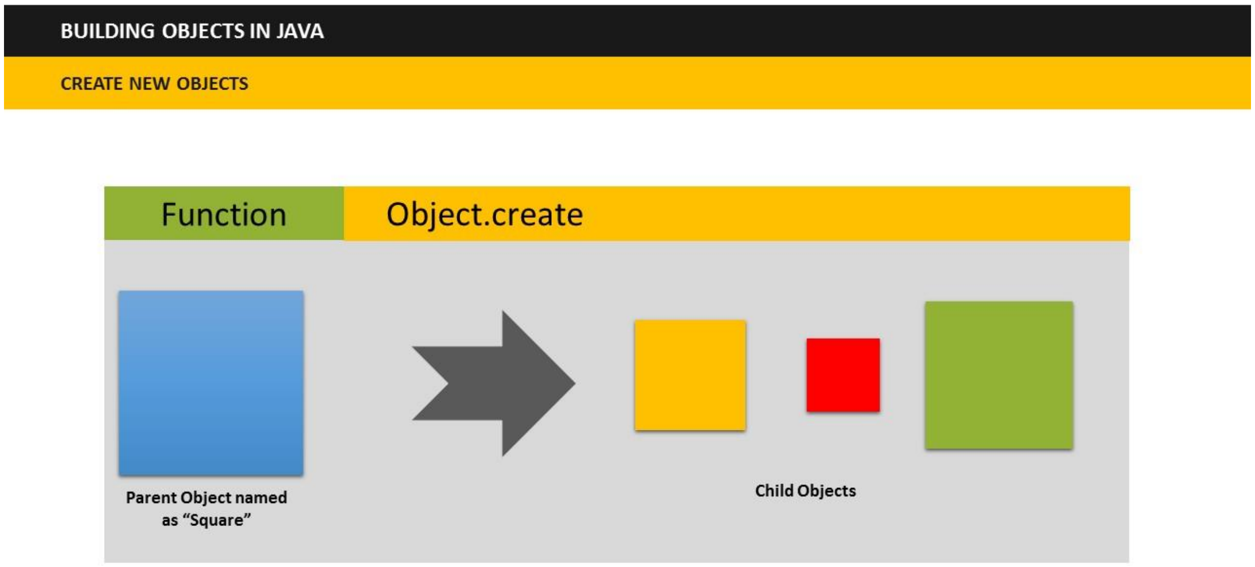


OBJECT-BASED PROGRAMMING CONCEPTS

WHAT ARE OBJECTS?

1. So, what are these objects? Every component in JavaScript is an object and these components include Functions, Strings, and Numbers. That means all the Functions, Strings, and Numbers can be called as objects in JavaScript.

## SUBTOPIC 1: CREATE NEW OBJECTS

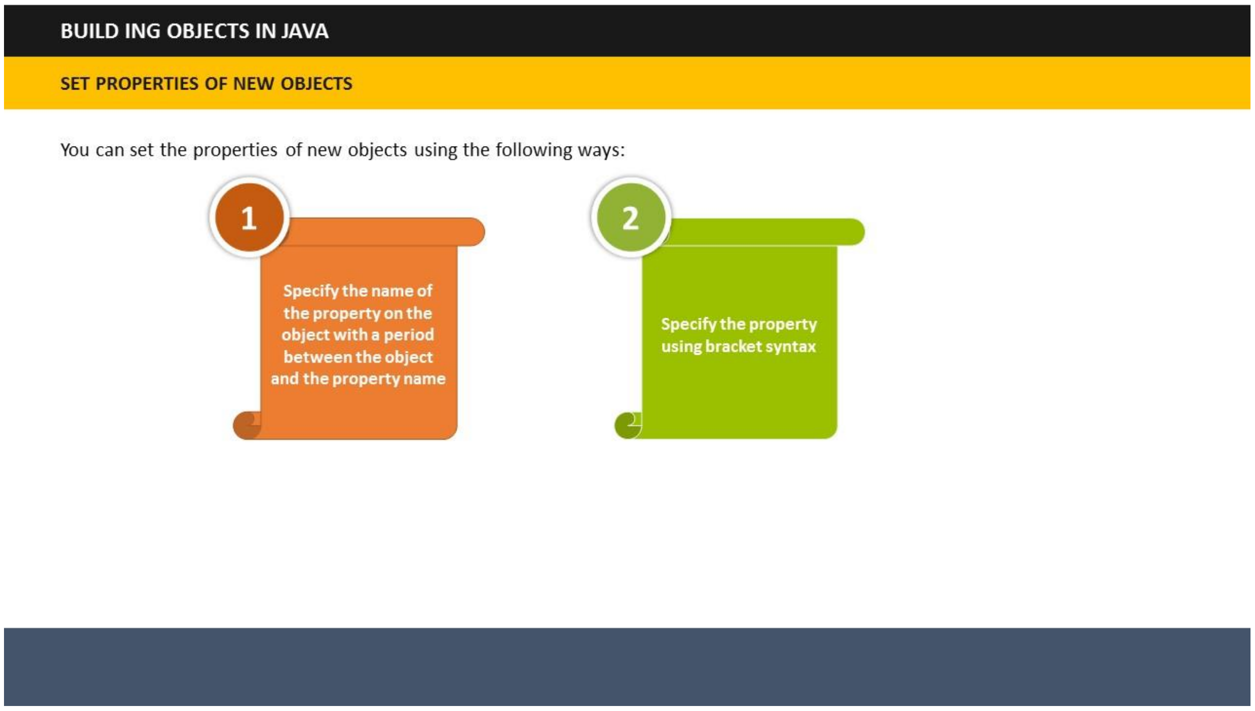


1. Now that you have an understanding of objects, let's see how we can create objects in JavaScript. This is quite simple.

2. The key to creating objects is the ***Object.create*** function. This is like creating child objects from a parent object as show on the screen. You can easily link the function to the name of the object from which you want to create your new object.

3. In the example shown on the screen, we can link the function with the parent blue square, which will define the child objects that are the orange, red and green squares.

## SUBTOPIC 2: SET PROPERTIES OF NEW OBJECTS



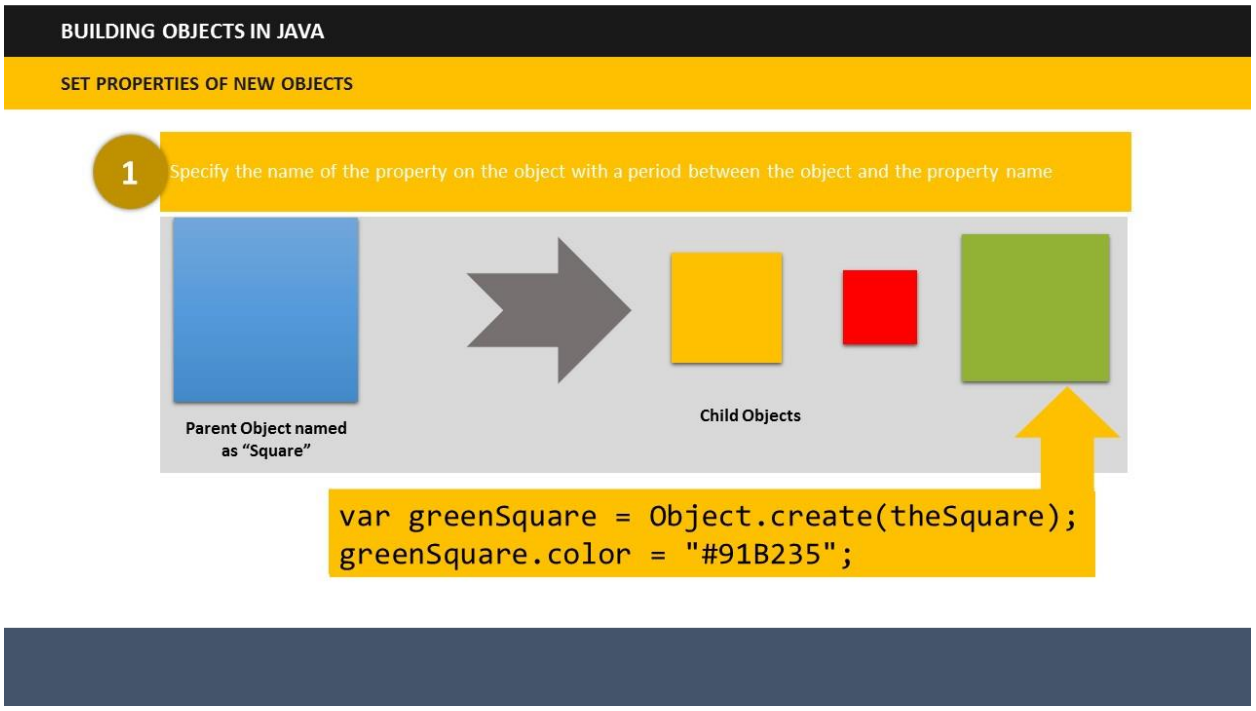
BUILD ING OBJECTS IN JAVA

SET PROPERTIES OF NEW OBJECTS

You can set the properties of new objects using the following ways:

1. Specify the name of the property on the object with a period between the object and the property name
2. Specify the property using bracket syntax

1. The next step is to set the properties of the newly created object.



BUILDING OBJECTS IN JAVA

SET PROPERTIES OF NEW OBJECTS

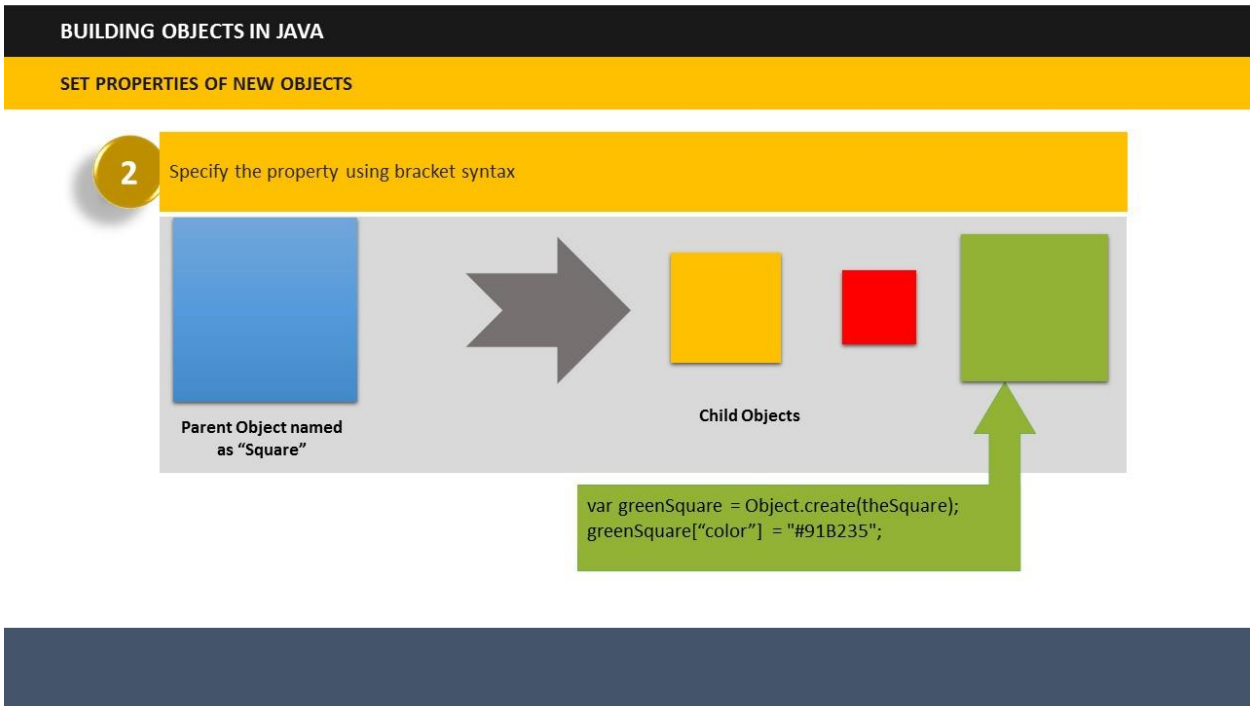
1 Specify the name of the property on the object with a period between the object and the property name

```
var greenSquare = Object.create(theSquare);
greenSquare.color = "#91B235";
```

2. Let’s take the example of parent and child objects that we just saw. So, if you want to set the color property of the green square using the first method, the script will look like this:

```
var greenSquare = Object.create(theSquare);
greenSquare.color = "#91B235";
```

3. Notice that we have set the color property on our green circle object to a value of #91B235.
4. The other method involve setting the property using bracket syntax. Let’s understand these two methods using examples.



BUILDING OBJECTS IN JAVA

SET PROPERTIES OF NEW OBJECTS

2 Specify the property using bracket syntax

```
var greenSquare  = Object.create(theSquare);
greenSquare["color"]  = "#91B235";
```

5. Next, let's see how we can set this green color using the second method of bracket syntax.

6. Notice that in the code the word "color" has been placed in square brackets. The end result will be the same irrespective of the method that is the square will be assigned a green color.

```
var greenSquare = Object.create(theSquare);
greenSquare["color"] = "#91B235";
```

## JavaScript classes

ES6 classes provide a much more familiar and structured way to implement OOP concepts for developers coming from languages like Java or C#.

### What is Object-Oriented Programming (OOP)?

OOP is a programming paradigm based on the concept of "objects," which can contain data (attributes or properties) and code (methods or functions) that operate on that data. The main goal of OOP is to increase the flexibility and maintainability of large, complex software systems.

The four pillars of OOP (often remembered by the acronym **A.P.I.E.** or **P.I.E.A.**) are:

1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**

Let's see how JavaScript classes help achieve these:

**1. Encapsulation**

**Concept:** Encapsulation is the bundling of data (properties) and the methods (functions) that operate on that data within a single unit (an object/class). It also involves controlling access to the internal state of an object, hiding its internal implementation details from the outside world.

**How Classes help in JavaScript:**

- **Bundling:** When you define a class, you naturally group related properties and methods within its body.

```
class BankAccount {
  constructor(accountNumber, balance) {
    this.accountNumber = accountNumber; // Data
    this.balance = balance;        // Data
  }

  deposit(amount) { // Method operating on data
    this.balance += amount;
    console.log(`Deposited ${amount}. New balance: ${this.balance}`);
  }

  withdraw(amount) { // Method operating on data
    if (this.balance >= amount) {
      this.balance -= amount;
      console.log(`Withdrew ${amount}. New balance: ${this.balance}`);
    } else {
      console.log("Insufficient funds.");
    }
  }
}

const myAccount = new BankAccount("123456789", 1000);
myAccount.deposit(500); // We interact through methods
// myAccount.balance = -100; // Directly accessing and changing can break logic
(problem addressed by private fields)
```

**Inheritance**

**Concept:** Inheritance allows a new class (subclass or child class) to inherit properties and methods from an existing class (superclass or parent class). This promotes code reusability and establishes a "is-a" relationship (e.g., a Dog *is a* Animal).

**How Classes help in JavaScript:**

The extends keyword is used for inheritance, and super() is used to call the parent class's constructor.

```
<html>
   <head>
      <title>ES6  Programming</title>
   </head>
     <body>
       <h1>JavaScript. Classes</h1>
       <script>   // TATA
             class Car{
                 brand="TATA";
                display(){
                  document.write("<br>My Car Name is :"+this.brand);
                }
             } // class close
             //car =new Car();
             // car.display();
           class Myclass  extends Car
           {       mydata(){
                document.write("<br>Myclass method");
              }
           }
            var myc =new Myclass();
                myc.mydata();
                myc.display();
                document.write("<br> My Brand is " +myc.brand);
        </script>
     </body>
</html>
```

**Polymorphism**

**Concept:** Polymorphism (meaning "many forms") allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different underlying forms (types). The most common form in OOP is **method overriding**, where a subclass provides its own implementation of a method that is already defined in its superclass.

**How Classes help in JavaScript:**

As seen in the Dog example above, Dog overrides the eat() method from Animal. When myDog.eat() is called, the specific Dog version is executed.

**Abstraction**

**Concept:** Abstraction is about hiding complex implementation details and showing only the essential features of an object. It focuses on "what" an object does rather than "how" it does it. In many languages, this is achieved through abstract classes or interfaces.

**Polymorphism**

**Concept:** Polymorphism (meaning "many forms") allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent different underlying forms (types). The most common form in OOP is **method overriding**, where a subclass provides its own implementation of a method that is already defined in its superclass.

**How Classes help in JavaScript:**

As seen in the Dog example above, Dog overrides the eat() method from Animal. When myDog.eat() is called, the specific Dog version is executed.

**Polymorphism example :**

```
<!DOCTYPE html>
<html>
<head>
  <title>Polymorphism Example (No Arrow Functions)</title>
</head>
<body>
  <h2>Polymorphism Demo</h2>
  <button id="demonstrateButton">Make Animals Eat</button>
  <div id="output"></div>

  <script>
```

```
// Utility functions
function log(message) {
  document.getElementById("output").innerHTML += message + "<br>";
}

function clearOutput() {
  document.getElementById("output").innerHTML = "";
}

// 1. Base Class: Animal
class Animal {
  constructor(name) {
    this.name = name;
  }
  eat() {
    log(this.name + " is eating generic food.");
  }
}

// 2. Subclass: Dog
class Dog extends Animal {
  constructor(name, breed) {
    super(name);
    this.breed = breed;
  }
  eat() {
    log(this.name + " (" + this.breed + ") devours its kibble!");
  }
  bark() {
    log(this.name + " barks loudly!");
  }
}

// 3. Subclass: Cat
class Cat extends Animal {
  constructor(name, color) {
    super(name);
    this.color = color;
  }
  purr() {
    log(this.name + " (" + this.color + ") purrs contentedly.");
  }
}

// Event Listener without arrow function
```

```
    document.getElementById("demonstrateButton").addEventListener("click", function()
{
      clearOutput();
      log("--- Demonstrating Polymorphism ---");

      var genericAnimal = new Animal("Lion");
      log("Calling eat() on a generic Animal object:");
      genericAnimal.eat();

      log("------------------------------------");

      var myDog = new Dog("Buddy", "Golden Retriever");
      log("Calling eat() on a Dog object:");
      myDog.eat();

      log("------------------------------------");

      var myCat = new Cat("Whiskers", "Tabby");
      log("Calling eat() on a Cat object (inherits eat()):");
      myCat.eat();

      log("------------------------------------");
      log("Notice how 'eat()' behaves differently for Dog vs. Animal/Cat.");
    });

    log("Click 'Make Animals Eat' to see the demo.");
  </script>
</body>
</html>
```