JavaScript
Functions

Naveen Pete

Functions

- Fundamental building block of JavaScript
- Allow you to store a piece of code that does a single task inside a defined block
- Call that code whenever you need it using a single short command
- Same as a Procedure or a Subroutine, in other programming languages

- To use functions, you need to
  - Define the function
  - Invoke (call) it

Functions

- Defining functions
  - A function definition consists of the ‘function’ keyword, followed by:
    - The name of the function
    - A list of parameters to the function, enclosed in parentheses and separated by commas
    - The JavaScript statements that define the function, enclosed in curly brackets, { }

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

Functions

- Calling functions
  - Defining a function does not execute it
  - Calling the function actually performs the specified actions with the indicated parameters
  - To call a function ‘greetUser()’ that takes a string parameter, we can use the following statement:

```
greetUser('Hari');
```

- Arguments
  - Function arguments are the values received by the function when it is invoked
  - Inside the function, the arguments (the parameters) behave as local variables

Functions

```
function square(num) {
  console.log(num * num);
}

square(10);
```

```
function area(length, width) {
  console.log(length * width);
}

area(5, 4);
```

```
function greet(person1, person2, person3) {
  console.log('hi ' + person1);
  console.log('hi ' + person2);
  console.log('hi ' + person3);
}

greet('hari', 'shiv', 'krish');
```

Functions

- Return value
  - Value returned by the function when it completes
  - Is optional, not all functions return a value
  - Generally, a return value is used where the function is an intermediate step in a calculation of some kind
  - We use the ‘return’ keyword to output a value from the function
  - The ‘return’ keyword stops execution of a function

```
function square(num) {
  return num * num;
}

var result = square(10);
console.log(result);
```

```
function capitalize(str) {
  if(typeof str === ‘number’) {
    return ‘Not a string’;
  }
  return str.charAt(0).toUpperCase()
         + str.slice();
}
```

Functions

- Function expression
  - A way to define a function inside an expression

```
var myFunction = function [name]([param[, param[, ..., param]]]) {
   statements
};
```

  - name
    - The function name. Can be omitted, in which case the function is anonymous. The name is only local to the function body.
  - param
    - The name of an argument to be passed to the function.
  - statements
    - The statements which comprise the body of the function.

Functions

- Function expression

```
var getRectArea = function(width, height) {
    return width * height;
}

console.log(getRectArea(3,4));
// expected output: 12
```

Functions

- Scope
  - Determines the accessibility (visibility) of variables
  - JavaScript has two scopes
    - Function scope (Local scope)
    - Global scope

  - Function scope
    - Each function creates a new scope
    - Variables defined within a function
      - are local to the function
      - can only be accessed within the function
      - are not visible (accessible) outside of the function
      - are created when the function starts and deleted when the function completes

Functions

- Scope
  - Global scope
    - Variables declared outside any function are global in nature
    - All functions and scripts have access to global variables
    - If you assign a value to a variable that is not declared, it will automatically become a global variable
- Global variables in HTML
  - In HTML, the global scope is the ‘window’ object
  - All global variables belong to ‘window’ object
  - The global scope is the complete JavaScript environment

Functions

- Higher order function
  - Function that takes a function as an argument (or)
  - Returns a function

Q & A

- Thank you!