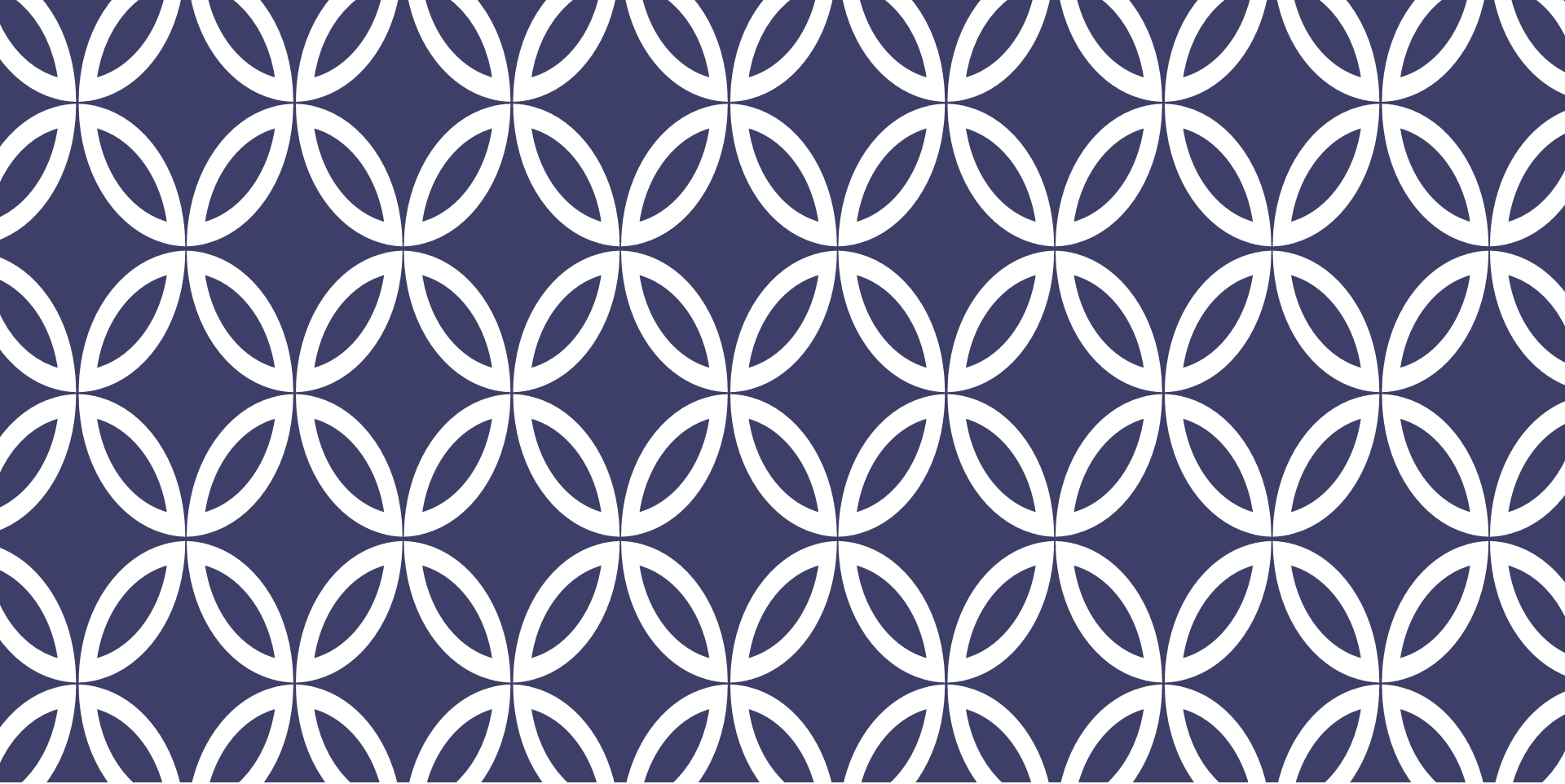


# ACADGILD

## Presents

### Front End Web Development Basics





# Session 6 – JavaScript



# Agenda – JavaScript

1. **JavaScript Identifiers**
2. **JavaScript Keywords**
3. **Conditional**
4. **if statement**
5. **if...else Statement**
6. **else if Statement**
7. **if/if ... else statement**
8. **switch statement**
9. **Looping statement**
10. **for loop**
11. **for –in loop**
12. **while loop**



# Agenda – JavaScript

- 11. do while loop**
- 12. function**
- 13. Accessing unnamed arguments**
- 14. Scope**
- 15. Debugging in JavaScript**



# JavaScript Identifiers

- **Identifiers** are used to name variables (and keywords, and functions, and labels)
- The rules for legal names are much the same in most programming languages
- First character must be a letter, an underscore (\_), or a dollar sign (\$)
- Subsequent characters may be letters, digits, underscores, or dollar sign
- Numbers are not allowed as the first character
- Reserved words (like JavaScript keywords) cannot be used as names
- All JavaScript identifiers are **case sensitive**.
- **Hyphens are not allowed** in JavaScript. It is reserved for subtractions.



# JavaScript Keywords

- JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.
  - **Break** Terminates a switch or a loop
  - **Continue** Jumps out of a loop and starts at the top
  - **Debugger** Stops the execution of JavaScript, and calls (if available) the debugging function
  - **do ... While** Executes a block of statements, and repeats the block, while a condition is true
  - **For** Marks a block of statements to be executed, as long as a condition is true
  - **Function** Declares a function
  - **if ... Else** Marks a block of statements to be executed, depending on a condition
  - **Return** Exits a function
  - **Switch** Marks a block of statements to be executed, depending on different cases
  - **try ... Catch** Implements error handling to a block of statements
  - **Var** Declares a variable



# Conditional

Conditionals are code structures that allow you to test whether an expression returns true or not, and then run different code depending on the result.

- “if” statement
- “if ... else” statement
- “else if” statement
- “if/if ... else” statement
- “switch” statement



# if Statement

- It is the main conditional statement in JavaScript.
- The keyword “if” always appears in lowercase.
- The condition yields a logical true or false value.
- The condition is true, statements are executed.

**Syntax** : `if (condition) { statements; }`





# if...else Statement

You can include an “else” clause in an if statement when you want to execute some statements if the condition is false.

## Syntax :

```
if (condition) { statements; }  
else { statements; }
```



## else if Statement

Allows you to test for multiple expression for one true value and executes a particular block of code.

### Syntax :

```
if (condition) { statement; }  
    else if (condition) { statement; }  
    else { statement; }
```



# if/if ... else statement

## Syntax :

```
if (condition) {  
    if (condition) { statements; }  
    else { statements; }  
}
```



# switch statement

Allows you to merge several evaluation tests of the same variable into a single block of statements.

## Syntax :

```
switch (expression) {  
    case label1:  
        statements; break;  
    default:  
        statements;  
}
```



# Looping Statement

Loops let you run a block of code a certain number of times.

- for” Loops
- “for/in” Loops
- “while” Loops
- “do ... while” Loops



# for loop

- One of the most used and familiar loops is the for loop.
- It iterates through a sequence of statements for a number of times controlled by a condition.
- The `change_exp` determines how much has been added or subtracted from the counter variable.

## Syntax :

```
for (initial_expression; test_exp; change_exp)
{ statements; }
```



## for –in loop

- When the for/in statement is used, the counter and termination are determined by the length of the object.
- The statement begins with 0 as the initial value of the counter variable, terminates with all the properties of the objects have been exhausted.
  - E.g. array → no more elements found

### Syntax :

```
for (counter_variable in object)  
{ statements; }
```



# while loop

- The while loop begins with a termination condition and keeps looping until the termination condition is met.
- The counter variable is managed by the context of the statements inside the curly braces.

## Syntax :

initial value declaration;

```
while (condition) {  
    statements;  
    increment/decrement statement;  
}
```





# do while loop

- The do/while loop always executes statements in the loop in the first iteration of the loop.
- The termination condition is placed at the bottom of the loop.

## Syntax :

initial value declaration;

```
while (condition) {  
    statements;  
    increment/decrement statement;  
}
```



# function

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses (**)**
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables)
- The parentheses may include parameter names separated by commas: **(parameter1, parameter2, ... So on )**
- The code to be executed, by the function, is placed inside curly brackets: **{ }**

## Syntax :

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



# Accessing Unnamed Arguments

## How do we get more arguments than listed in parameters?

- There is a special pseudo-array inside each function called **arguments**.
- It contains all parameters by their  
number: arguments[0], arguments[1] etc.

- **Example :**

```
function sayHi() {  
  for(var i=0; i<arguments.length; i++) {  
    alert("Hi, " + arguments[i])  
  }  
  sayHi("Ron", "Alice") // 'Hi, Ron', then 'Hi, Alice'
```



# Scope

**“Scope”** refers to the variables that are available to a piece of code at a given time.

**Functions have access to variables defined in the same scope**

**Example :**

```
var foo = 'hello';
```

```
var sayHello = function() {  
  console.log(foo);  
};
```

```
sayHello(); // logs 'hello'  
console.log(foo); // also logs 'hello'
```



# Scope

Code outside the scope in which a variable was defined does not have access to the variable

## Example :

```
var sayHello = function() {  
  var foo = 'hello';  
  console.log(foo);  
};
```

```
sayHello(); // logs 'hello'  
console.log(foo); // doesn't log anything
```



# Scope

Variables with the same name can exist in different scopes with different values

## Example :

```
var foo = 'world';
```

```
var sayHello = function() {  
  var foo = 'hello';  
  console.log(foo);  
};
```

```
sayHello(); // logs 'hello'  
console.log(foo); // logs 'world'
```



# Debugging in JavaScript

With the recent boom of JavaScript, all major browsers come with their own debug tools.

## Examples:

- Chrome has [Chrome DevTools](#). You can access it by shortcut key Ctrl+Shift+Alt.
- For Firefox you can use [firebug extension](#).



# Lets Discuss Assignments