

THE INTRODUCTION TO

JAVASCRIPT PART 2

RECAP

- ▶ What is the purpose of JavaScript on the Web?
- ▶ What are the five primitive data types?
- ▶ What are Strings?
- ▶ What are Numbers?
- ▶ What are Booleans?
- ▶ What are Variables?
- ▶ What are Comments?

OUTLINE

- ▶ Operators
- ▶ Conditional Statements
- ▶ Loops
- ▶ Arrays
- ▶ Functions

OPERATORS

- ▶ Operators are the symbols between values that allow different operations like addition, subtraction, multiplication... etc.
- ▶ Common Operators:
 - ▶ Arithmetic Operators:
 - + The plus operator adds two numbers
And concatenates strings together.

```
10 + 5 //outputs 15
```

```
"Hello" + " " + "World" //outputs Hello World
```

OPERATORS

▶ Arithmetic Operators (continued)

- The - operator subtracts one number from another.

- * The * operator multiplies two numbers.

(Notice this is a * and not an x)

- / The / operator divides one number by another.

▶ JavaScript expressions follow an order of operations.

▶ Order of operations can be controlled by the Grouping operator

() The () operator groups other values and operations.

Example: `(5 + 100) * 5 // outputs 525`

▶ Assignment operator

= The = operator assigns values. Used in setting the values of variables.

OPERATORS

► Comparison Operators

== This is the equal to operator

Example: `var myNumber = 10;`

```
11 == myNumber; //outputs false
```

```
10 == myNumber; //outputs true
```

!= This is the not equal to operator

Example: `12 != myNumber; //outputs true`

> This is the greater than operator

Example: `100 > myNumber //outputs true`

OPERATORS

► Comparison operators (continued)

< This is the less than operator.

Example: `100 < myNumber //output false`

>= This is the greater than or equal to operator.

Example: `120 >= myNumber //outputs true`

<= This is the less than or equal to operator.

Example: `10 <= myNumber //outputs true`

OPERATORS

- ▶ Logical operators are used to determine the logic between variables or values.

&& This is the and operator

Example:

```
(mynumber < 100 && mynumber == 10) //outputs true
```

|| This is the or operator

Example:

```
(mynumber >= 200 || mynumber > 5) //outputs true
```

! This is the not operator

Example:

```
!(10 == myNumber) //outputs false
```

```
!(13 < myNumber) //outputs true
```


JAVASCRIPT CONDITIONAL STATEMENTS

- ▶ Conditionals control behavior in JavaScript and determine whether or not a blocks or pieces of code can run.

The most common type of conditional statement is the **if** statement, which only runs if the condition enclosed in parentheses () is **truthy**.

```
if(myNumber < 100) {  
    console.log("myNumber is less than 100");  
}
```

▶ else

You can extend an if statement with an else statement, which adds another block to run when the **if** conditional doesn't pass.

```
if(myNumber < 100) {  
    console.log("myNumber is less than 100");  
} else {  
    console.log("myNumber is greater than 100");  
}
```

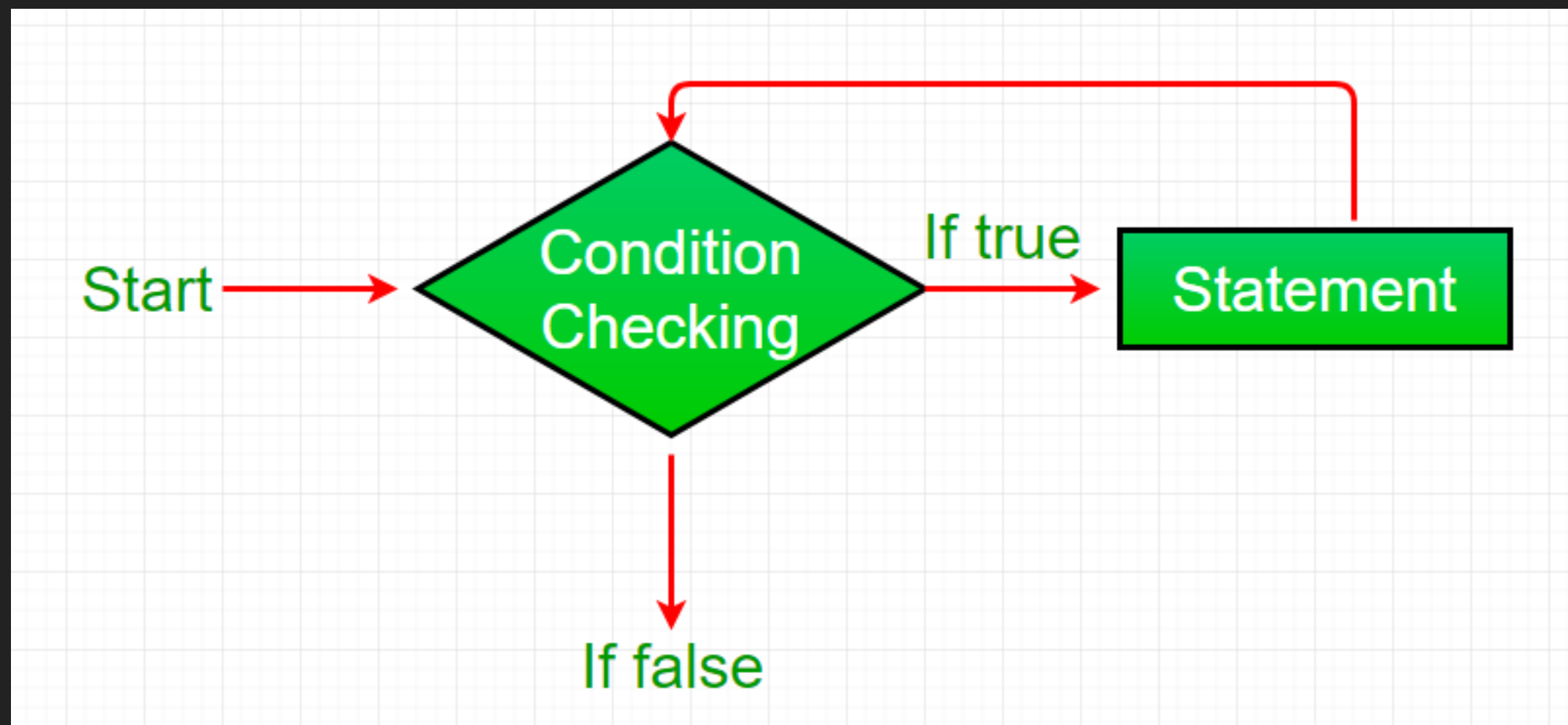
▶ else if

You can also extend an `if` statement with an `else if` statement, which adds another conditional with its own block.

```
if(myNumber < 100) {  
    console.log("myNumber is less than 100");  
} else if(myNumber == 20) {  
    console.log("myNumber is equal to 20");  
} else {  
    console.log("myNumber is not equal to 20 or less than 100")  
}
```

JAVASCRIPT LOOPS

- ▶ Looping is a feature in JavaScript and many other programming languages that facilitates the execution of instructions until the condition is true.



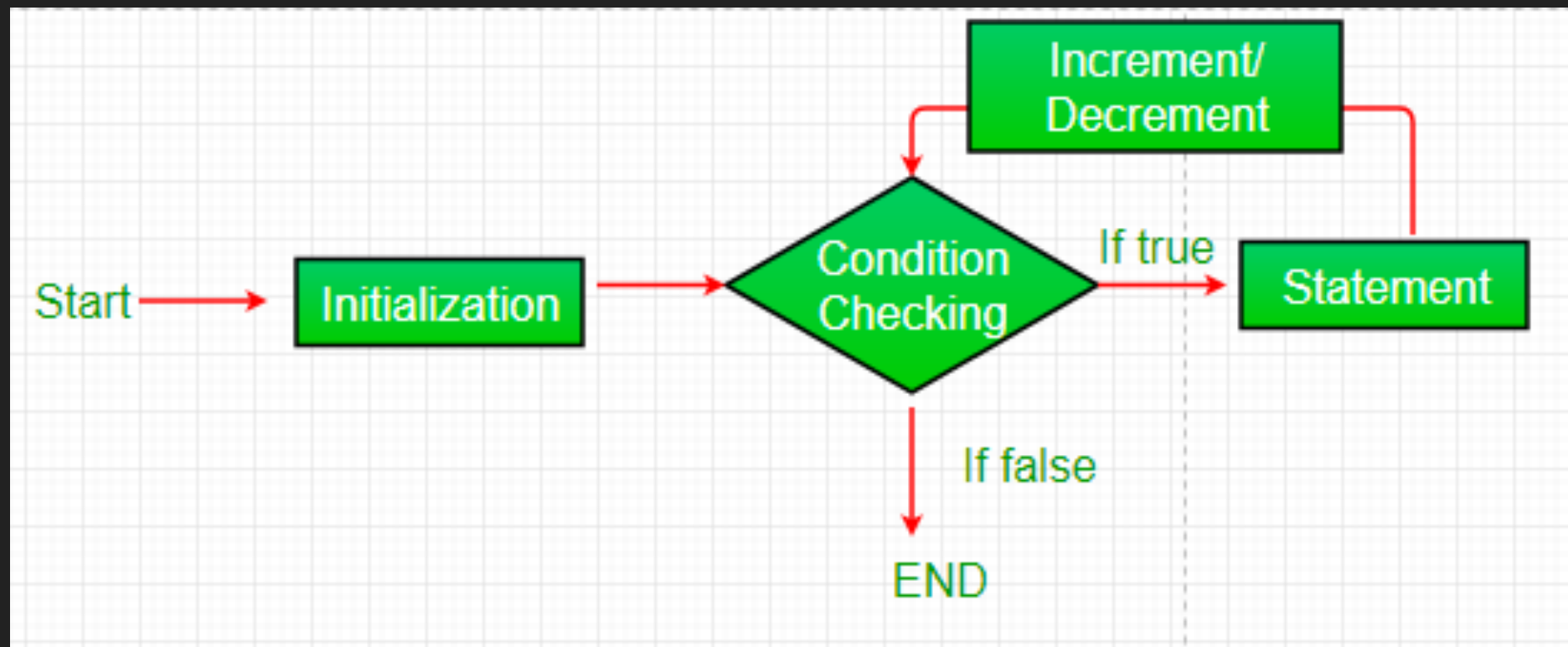
► while

A while loop is a control flow statement that allows code to be executed repeatedly base on a give Boolean condition. The while loop can be thought of as a repeating if statement.

```
while(myNumber < 20) {  
    console.log("I love puppies");  
    myNumber = myNumber + 1;  
}
```

► for

A for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition, and increment/decrement in one line, providing a shorter, easy to debug structure of looping.



- ▶ for loop continued:
 - ▶ First you initialize the condition by making a variable to use. It marks the start of the loop. You can use a variable that is already declared or create one that will be local to the loop.
 - ▶ Second - you set a testing condition.
 - ▶ Third - if the testing condition is true, the statement in the loop body executes.
 - ▶ Fourth - the increment or decrement is used for updating the variable for the next iteration.
 - ▶ Fifth - when the condition becomes false, the loop ends marking the end of the cycle.

- ▶ A common practice with for loops is to create a variable named `i` (which is shorthand for iterator) and to use the shorthand addition and subtraction operators.

```
i++; //is equal to the statement i = i + 1;  
i--; //is equal to the statement i = i - 1;
```

```
for (var i = 0; i < 200; i++) {  
    console.log("i is equal to " + i );  
}
```


JAVASCRIPT ARRAYS

- ▶ Arrays are container-like values that can hold other values. The values inside an array are called elements.
- ▶ When creating an array the common practice is to create a variable and use the [] brackets to initialize the array.
Example:

```
var namesArray = ["Mr. Fancy", "Da Business", "Bunkis"]
```

elements are separated by a comma

JAVASCRIPT ARRAYS

- ▶ Array elements don't all have to have the same type of values. Elements can be any kind of JavaScript value, including an array.

```
var multiValueArr = ["String", [3,44,2],2000,false,{key:"value"}]
```

- ▶ To access an array element you'll use the brackets with a specific element number inside.

Example: `multiValueArr[2];` *//output is 2000*

Arrays elements begin at 0.

JAVASCRIPT ARRAYS

- ▶ To access the last element in an array we can use the `.length` property. But wait! The `.length` property starts counting at 1 not 0. So to access the last element in an array we subtract 1.

Example:

```
multiValueArr[multiValueArr.length - 1]; //output is {key:"value"};
```

- ▶ We can also reassign elements by using bracket notation.

Example: `multiValueArr[0] = "Captain Fancy";`

- ▶ Functions are blocks of code that can be named and reused.

Example:

```
function sayHello(name) {  
    return "Hello " + name + "!";  
}
```

1. We use the function keyword to declare a function.
2. We set a name (sayHello) - make it declarative.
3. We set parameters that the function will accept.
4. We use the return keyword that exits the function and outputs the value.

JAVASCRIPT FUNCTIONS

- ▶ Using or invoking functions is accomplished by referencing its name followed by () parenthesis right after, and any parameters inside the ();

Example:

```
sayHello("Captain Fancy") //output Hello Captain Fancy!
```

- ▶ Mozilla Developer Network is the official Mozilla website for development documentation of web standards and Mozilla projects. A huge resource to look into specific JavaScript functionality <https://developer.mozilla.org/en-US/>
- ▶ <https://repl.it> is a tool that allows you to run simple programs written in multiple languages including JavaScript with an integrated console.
- ▶ <https://stackoverflow.com/> - **Stack Overflow** is a question and answer site for professional and enthusiast programmers. If you have questions about a programming pattern, or problem - this site is an invaluable tool that professional software engineers use everyday.