JavaScript
Control Flow

Naveen Pete

Operators

- Comparison Operators
  - Let us assume that x = 5

Operator	Name	Example	Result
>	Greater than	x > 10	false
>=	Greater than or equal to	x >= 5	true
<	Less than	x < -50	false
<=	Less than or equal to	x <= 100	true
==	Equality	x == “5”	true
!=	Inequality	x != ‘b’	true
===	Strict equality	x === “5”	false
!==	Strict inequality	x !== “5”	true

  - == vs ===
    - == performs type coercion, while === does not

Operators

```
true == “1”
0 == false
null == undefined
NaN == NaN
```

- Logical Operators
  - Let us assume that x = 5 and y = 9

Operator	Name	Example	Result
&&	AND	x < 10 && x !== 5	false
||	OR	y > 9 || x === 5	true
!	NOT	!(x === y)	true

Operators

- Truthy and Falsy
  - A truthy value is a value that is considered true when evaluated in a Boolean context
  - A falsy value is a value that translates to false when evaluated in a Boolean context

  - Following are Falsy values:
    - false
    - 0
    - “” or ‘’
    - null
    - undefined
    - NaN
    - document.all
  - Everything else is Truthy

Operators

```
var x = 10;
var y = “a”;

y === “b” || x >= 10;
```

```
var x = 3;
var y = 8;

!(x == “3” || x === y) && !(y != 8 && x <= y);
```

```
var str =“”;
var msg = “hi”;
var isFalse = “false”;

!((str || msg) && isFunny);
```

Conditionals

- Is a way to add decisions to your code
- Allows you to add logic to your programming
  - E.g. user log on, bank transaction

- The if statement executes a statement if a specified condition is truthy. If the condition is falsy, another statement can be executed

```
if (condition)
   statement1
[else
   statement2]
```

  - ***condition*** - An expression that is considered to be either truthy or falsy
  - ***statement1*** - Statement that is executed if condition is truthy. Can be any statement, including further nested if statements
  - ***statement2*** - Statement that is executed if condition is falsy and the else clause exists

Loops

- DRY – Don’t Repeat Yourself
  - Saves us time
  - Makes our code cleaner

- while loop
  - The while statement creates a loop that executes a block of code as long as the test condition evaluates to true
  - The condition is evaluated before executing the statement

```
while (condition) {
    code block to be executed
}
```

Loops

- do…while loop
  - The do...while statement creates a loop that executes a block of code until the test condition evaluates to false
  - The condition is evaluated after executing the statement, resulting in the specified statement executing at least once

```
do {
    code block to be executed
}
while (condition);
```

  - ***Note***
    - Infinite loops occur when the terminating condition in a loop is never false

Loops

- for loop
  - The for statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement (usually a block statement) to be executed in the loop

```
for (statement-1; statement-2; statement-3) {
    code block to be executed
}
```

  - ‘statement-1’ is executed (one time) before the execution of the code block
  - ‘statement-2’ defines the condition for executing the code block
  - ‘statement-3’ is executed (every time) after the code block has been executed

Q & A

- Thank you!