# Internet Programming

## Lecture 02

Instructor:
Daniel Slack, P. Eng.
Applied Computer Science
University of Winnipeg

# What is JavaScript?

- JavaScript was released in 1995
  - A way to add programs to web pages, first in the Netscape Navigator browser
  - Has since been adopted by all other major web browsers
  - It has made modern web applications possible
    - Support in browsers means no extra plugins necessary
    - Applications you can interact with directly

# What is JavaScript?

- JavaScript has nothing to do with the Java programming language
- The similar name was a marketing ploy
  - At the time, Java was heavily marketed as the "hot new" web programming language
  - Java applets were going to change the web
  - We are stuck with the name
    - It officially is an implementation of the ECMAScript standard

# Values, Types, and Operations

- Inside the computer's world, there is only data
- You can read data, modify data, create new data
  - Anything that isn't data simply does not exist
  - All data is stored as long sequences of bits
    - Fundamentally it is alike.

# Values

- To be able to work with, you can separate them into chunks that represent pieces of information
- In a JavaScript environment, those chunks are called *values*
  - Though all values are made of bits, they play different roles.
  - Every value has a type that determines its role
- There are six basic types of values in JavaScript:
  - numbers, strings, booleans, objects, functions, and undefined values.

# Numbers

- Values of the *number* type are numeric values
- JavaScript uses 64 bits to store a single number value
  - There are only so many patterns you can make with 64 bits
    - Patterns are limited
    - For *N* decimal digits, the amount of numbers that can be represented is $10^N$
    - Given 64 binary digits, you can represent $2^{64}$ different numbers

# Special Numbers

- Three special values in JavaScript that are considered numbers
  - Don't behave like normal numbers
- First two are Infinity and -Infinity
  - Represent the positive and negative infinities
  - Infinity - 1 is still Infinity, and so on
- The other is NaN
  - NaN stands for "not a number"
  - You'll get this result when you try to calculate:
    - 0 / 0 (zero divided by zero)
    - Infinity - Infinity, or any number of other numeric

# Strings

- Strings are used to represent text
- Both single and double quotes can be used to mark strings
  - As long as the quotes at the start and the end of the string match

# Boolean Values

- JavaScript has a *Boolean* type, which has just two values true and false
  - Written simply as those words

# Logical Operators

- There are also some operations that can be applied to Boolean values themselves
  - JavaScript supports three logical operators:
    - *and*, *or*, and *not*
    - These can be used to "reason" about Booleans.
- && || !
- Note:
  - The expression to their right is evaluated only when necessary

# Undefined Values

- There are two special values, written null and undefined
  - Are used to denote the absence of a meaningful value
  - They are themselves values, but they carry no information.

# Aside: Console.log

- We will use console.log in example code to indicate that we want to see the result of evaluating something
- When you run such code, the value produced should be shown on the screen
  - How it appears will depend on the JavaScript environment you use to run it

# Unary Operators

- Not all operators are symbols
  - Some are written words
- One example is the *typeof* operator
  - Produces a string value naming the type of the value you give it

```
console.log(typeof 4.5)
console.log(typeof “x”)
```

# Automatic Type Conversion

- JavaScript goes out of its way to accept almost anything you give it
  - Even programs that do odd things

```
console.log(8 * null)
// → 0
console.log("5" - 1)
// → 4
console.log("5" + 1)
// → 51
console.log("five" * 2)
// → NaN
console.log(false == 0)
// → true
```

# Automatic Type Conversion

- When an operator is applied to the "wrong" type of value, JavaScript will quietly convert that value to the type it wants
  - Using a set of rules that often aren't what you want or expect
  - This is called *type coercion*
- When something that doesn't map to a number in an obvious way is converted to a number
  - The value NaN is produced

# Automatic Type Conversion

- When comparing values of the same type using ==, the outcome is easy to predict:
  - You should get true when both values are the same
    - Except in the case of NaN
  - But when types differ, JavaScript uses a complicated and confusing set of rules to determine what to do
    - Usually, just tries to convert one of the values to the other value's type
    - However, when null or undefined occurs on either side of the operator, it produces true only if both sides are one of null or undefined

```
console.log(null == undefined);
// → true
console.log(null == 0);
// → false
```

# Automatic Type Conversion

- That last piece of behavior is often useful
  - Can compare unknown value to null when you want to test whether it has a real value instead of null or undefined
- What if you want to test whether something refers to the precise value *false*?
  - 0, NaN, and "" count as false
    - All other values count as true
    - All true: 0 == false and "" == false

# Automatic Type Conversion

- There are extra operators ror cases where you do not want any automatic type conversions
  - === and !==
  - ===
    - Tests whether a value is precisely equal to the other
  - !===
    - Tests whether it is not precisely equal
    - False: “” === false