Data Types & Operations

Lesson Objectives

- Understand datatypes
- Understand arithmetic, relational and logical operators.
- Understand implicit and explicit type conversions.
- Learn to use Math object.

Data Types

- All programing languages have idea of data types, some languages are strict on data types, and others are loose.
- In JavaScript, a variable gets its type based on the value it is currently holding.
- There are 6 primitive data types: **`string`**, **`number`**, `bigint`, **`boolean`**, **`undefined`**, and `symbol`.
  - There also is **`null`**, which is seemingly primitive, but indeed is a special case for every **`Object`**.
- We can put any type in a variable.

```
// no error
let message = "hello";
message = 123456;
```

- Programming languages like this are called *“dynamically typed”*

Number

- The *number* type represents both integer and floating-point numbers.
- There are many operations for numbers, e.g., multiplication *, division /, addition +, subtraction -, and so on.
- Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: `Infinity`, `-Infinity` and `NaN`.

```
let n = 123;
n = 12.345;
console.log( 1 / 0 ); // Infinity
console.log( "not a number" / 2 ); // NaN
```

- `BigInt` type was added to the language to represent integers of arbitrary length. (reading)

String

- A string in JavaScript must be surrounded by quotes.

```
let str = "Hello";
let str2 = 'Single quotes are ok too’;
let phrase = `can embed another ${str}`;
```

- There is no char type
- A string may consist of zero characters (be empty), one character or many of them.
- Double or single quotes allows
  - We will favor double quotes

Boolean (logical type)

- The `boolean` type has only two values: `true` and `false`.

```
let isLazy = false;
let isHealthy = true;
```

null and undefined

- The special values `null` and `undefined` are special types as well.
- In JavaScript, `null` is not a “reference to a non-existing object” or a “null pointer” like in Java
  - special value which represents “nothing”, “empty” or “value unknown”.
- The meaning of `undefined` is “value is not assigned”.
  - If a variable is declared, but not assigned, then its value is `undefined`.
- Programmers assign null; the compiler assigns undefined

```
let name = null;
let age;
console.log(name,age) // null, undefined
```

Objects and Symbols

- Object is a complex data type.
  - Objects are used to store collections of data and more complex entities

- All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever)

The `typeof` operator

- The `typeof` operator returns the type of the argument.

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
```

Interactions: alert, prompt, confirm

- In browser environment, there are inbuild **functions** to interact with the user `alert`, `prompt` and `confirm`.
- In Node.js environment we will be using external `prompt-sync` module for input from the console.

```
const prompt = require("prompt-sync")();

let name = prompt("What is your name?: ")
console.log(`Hi ${name}`)
```

Exercise

- Create a folder for today's work, w1d2dataTypes
- Add a file, prompt.js
- add the following code and run the file
  - First need to install this function in node for the prompt
  - npm i prompt-sync

```
const prompt = require("prompt-sync")();

let name = prompt("What is your name?: ")
console.log(`Hi ${name}`)
```

Type Conversions

- Most of the time, operators and functions automatically convert the values given to them to the right type.
  - For example, `alert` automatically converts any value to a string to show it.
  - Mathematical operations convert values to numbers.

See example: lecture_codes/lesson2/implicit_type_conversion.js

- There are also cases when we need to explicitly convert a value to the expected type.

String Conversion

- String conversion happens when we need the string form of a value.
  - For example, `alert(value)` does it to show the value.
- We can also call the String(value) function to convert a value to a string:

```
let b = true;
let n = 5;

console.log(typeof b, typeof n) // boolean number

let s1 = String(b);
let s2 = String(n);

console.log(typeof s1, typeof s2) // string string
```

Numeric Conversion

- Numeric conversion happens in mathematical functions and expressions automatically.
  - For example, when division / is applied to non-numbers:

```
alert( "6" / "2" ); // 3, strings are converted to numbers
```

  - We can use the Number(value) function to explicitly convert a value to a number:

```
let str = "123.33";
let num = Number(str); // becomes a number 123.33
num = parseFloat(str); // becomes a number 123.33
num = parseInt(str); // becomes a number 123
```

  - Favor Number unless have a specific need for parseInt/parseFloat

Numeric conversion rules

Value	Becomes…
undefined	NaN
null	0
true and false	1 and 0
string	Whitespaces from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is “read” from the string. An error gives NaN.

```
console.log( Number("   123   ") ); // 123
console.log( Number("123z") );      // NaN (error reading a number at "z")


console.log( Number(true) );        // 1
console.log( Number(false) );       // 0
```

User input may need numeric conversion

- User input always comes as string, even when user may have entered a number.
- Before arithmetic operation can be performed, string should be converted to numeric type.
  - `Number(string)`
  - `parseInt(string)`
  - `parseFloat(string)`
  - *`Using unary (+) operator (reading)`*

- See example: lesson2/parsing_user_input.js

Boolean Conversion

- Boolean conversion is the simplest one.
  - It happens in logical and relational operations (covered later)
  - But can also be performed explicitly with a call to Boolean(value).
- The conversion rule:
  - Values that are intuitively “empty”, like 0, an empty string, null, undefined, and NaN, become false.
  - Other values become true.

```
console.log( Boolean(1) ); // true
console.log( Boolean(0) ); // false

console.log( Boolean("hello") ); // true
console.log( Boolean("") ); // false
```

Operations & Operators

- Different set of operations can be performed on a variable based on its data type
  - Arithmetic operations use operators (+, -, *,**, /, %, ++, --)
    - *When at least one of the operand type is string, + operator will perform string concatenation.*
  - Comparisons use relational operators (==, !=, >, <, >=, <=)
  - Logical operations use operators (&&, ||, !)

Arithmetic Operations

- These are similar operations as in mathematics (algebra)
  - same precedence
- modulus operator (%) returns remainder
- division (/) operator returns quotient.
- Multiplication (*) operator must be explicitly used
- exponentiation operator a ** b multiplies a by itself b times.
- What is result of following arithmetic operations?
  - 2-9+8-6+5
  - 2-9+8-6*5
  - parens always a good idea to remove any ambiguity

Exercise

- Following program asks user to input temperature value in degree Celsius and outputs the result in degree Fahrenheit. Make this program run on your machine.

```
let prompt = require('prompt-sync')();
let tempInCelsius = prompt('Eneter value in celsius: ')
let tempInFahrenheit = 9/5*parseFloat(tempInCelsius)+32;
console.log(tempInFahrenheit);
```

- Now write a program that asks a user to input temperature value in degree Fahrenheit and outputs the result in degree Celsius.

More assignment operators

- These are merely shortcuts

Operator	Example	Equivalent
+=	x += 2	x = x+2

- Same rule for -, *, / and % operators

Increment and Decrement Operators

- Shortcuts to increment and decrement current value by 1

  ++ count; => count += 1; => count = count + 1;

  -- count; => count -= 1; => count = count – 1;

- Pre vs Post, increment and decrement

  ++ count vs count ++

  -- count vs count –

  - Avoid using these in expressions (and in general)

- See example: lecture_codes/lesson2/increment_decrement_operators.js

Operator Precedence Revisited

Operator(s)	Name(s)
`()`	parentheses
`- (unary)`	negation
`++  --`	increment, decrement
`*  /  %`	multiplication, division, modulus
`+  -`	additions, subtraction
`=  +=  -=  *=  /=  %=`	assignments

Math Object

- Built-in object part of JavaScript language
  - functionality for mathematical computations

- See example: lecture_codes/lesson2/math.js

Exercise

- Write a program that computes volume of a cylinder based on user inputs for radius and height of a cylinder, using formula $v = \pi r^2 h$

- Write a program that takes x and y co-ordinates of two points as inputs and computes distance between these two points based on the formula, $d = \sqrt{(x 2 - x 1)^2 + (y 2 - y 1)^2}$

Relational operators

- As the name implies, these operators compare two values and return either true or false (Boolean values).
- <, >,<=, >=, ==, ===, !=, !==

- Strings in JavaScript are compared based on so-called “dictionary” or “lexicographical” order.

- See example: lecture_codes/lesson2/relational_operators.js

Comparing different types

- When comparing values of different types, JavaScript converts the values to numbers.

```
alert( '2' > 1 ); // true, string '2' becomes a number 2
alert( '01' == 1 ); // true, string '01' becomes a number 1
```

- A strict equality operator === checks the equality without type conversion.
  - In other words, if a and b are of different types, then a === b immediately returns false without an attempt to convert them.
  - Always use === instead of ==

- Generally, comparison of different types is a mistake or poor design

Logical Operators

- Logical operators (&&, ||, !), usually used with Boolean (logical) values.
  - Although they are called “logical”, they can be applied to values of any type, not only boolean.
  - *Their result can also be of any type (see slide on short-circuit evaluation).*

- Recall, relational operators return Boolean values.
  - logical operators can be to used to combine two or more relational expressions.

- See example: lecture_codes/lesson2/logical_operators.js

Short-circuit evaluation

- The && and || operators actually return the value of one of the specified operand, so if these operators are used with non-Boolean values, they will return non-Boolean value.
  - See example: lecture_codes/lesson2/short_circuit_or.js
  - Avoid doing this practice (unless highly experienced, and even then)

Truth Tables

Exp1	Exp2	Exp1 && Exp2	Exp1 || Exp2
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

Exp	!Exp
F	T
T	F

Truthy & Falsy

- In JavaScript, any value can be used as a Boolean
  - "**falsy**" values: 0, 0.0, NaN, "", null, and undefined
  - "**truthy**" values: anything else

Main point

- Data types determine the operations that can be carried out on data. The same operator can do different operations depending on the data type. *Science of consciousness, transcending is one operation that every human nervous system is capable of. TM operates in same way for every human nervous system.*

Assignments

- Readings
  - References from next slide
  - Statements vs Expressions | Programming.Guide
- See document provided in the Sakai Resources > assignments folder

- Upload your assignments to your GitHub repository and submit your status report as per the standard instructions.

References

- Data types (javascript.info)
- Interaction: alert, prompt, confirm (javascript.info)
- Type Conversions (javascript.info)
- Basic operators, maths (javascript.info)
- Comparisons (javascript.info)
- Logical operators (javascript.info)
- Nullish coalescing operator '??' (javascript.info)
- Operator precedence - JavaScript | MDN (mozilla.org)