INLEDANDE WEBBPROGRAMMERING MED JAVASCRIPT

INTRODUCTION TO WEB PROGRAMING USING JAVASCRIPT

ME152A
L1: VALUES, TYPES, VARIABLES & OPERATORS

OUTLINE

- Why programing languages are needed.
- Values, Types and operators
  - Numbers
  - Arithmetic operators
  - Assignment operators
  - Strings
  - Boolean values
  - Logical operators
- Variables
- Vectors

WHY PROGRAMING LANGUAGE?

- In the beginning programs looked like this:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

WHY PROGRAMING LANGUAGE? CONT'D

- In English can be written:

```
1. Store the number 0 in memory location 0.
2. Store the number 1 in memory location 1.
3. Store the value of memory location 1 in memory location 2.
4. Subtract the number 11 from the value in memory location 2.
5. If the value in memory location 2 is the number 0,
   continue with instruction 9.
6. Add the value of memory location 1 to memory location 0.
7. Add the number 1 to the value of memory location 1.
8. Continue with instruction 3.
9. Output the value of memory location 0.
```

Using names instead of numbers:

```
 Set "total" to 0.
 Set "count" to 1.
[loop]
 Set "compare" to "count".
 Subtract 11 from "compare".
 If "compare" is zero, continue at [end].
 Add "count" to "total".
 Add 1 to "count".
 Continue at [loop].
[end]
 Output "total".
```

WHY PROGRAMING LANGUAGE? CONT'D

- JavaScript example, the while construct:

```
1 var total = 0, count = 1;
2 while (count <= 10) {
3   total += count;
4   count += 1;
5 }
6 console.log(total);
7 // → 55
```

- If convenient operations "range" and "sum" available:

```
1 console.log(sum(range(1, 10)));
2 // → 55
```

VALUES, TYPES, AND OPERATORS

- Six basic types of values in JavaScript:
  - numbers,
  - strings,
  - Booleans,
  - objects,
  - functions, and
  - undefined values.

- Values and types are the basis for programming
- All values are a type

NUMBERS

- Values of the *number* type are, numeric values, example :
  - Example: 13
- Fractional numbers are written by using a dot,
  - Example: 9.81
- For very big or very small numbers, you can also use scientific notation by adding an “e” (for “exponent”), followed by the exponent of the number:
  - Example: 2.998e8 ($2.998 \times 10^8 = 299{,}800{,}000.$)

- **NOTE: consider fractional digital numbers as approximations, not as precise values in comparing to “Integers” (or whole numbers), which are always precise.**

ARITHMETIC

100 + 4 * 11

The + and * symbols are called *operators*.

+: addition, *: multiplication.

(100 + 4) * 11

For subtraction, there is the - operator, and division can be done with the / operator.

What would be the output of these two examples?

JAVASCRIPT ARITHMETIC OPERATORS

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

http://www.w3schools.com/js/js_operators.asp

SPECIAL NUMBERS

There are three special values in JavaScript that are considered numbers but don't behave like normal numbers.

Infinity and -Infinity,

which represent the positive and negative infinities.

NaN stands for "not a number"

JAVASCRIPT ASSIGNMENT OPERATORS

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

http://www.w3schools.com/js/js_operators.asp

JAVASCRIPT ASSIGNMENT OPERATORS: ADDITION EXAMPLE

```
// Assuming the following variables
//  foo = "foo"
//  bar = 5
//  baz = true


// Number + Number -> addition
bar += 2 // 7

// Boolean + Number -> addition
baz += 1 // 2

// Boolean + Boolean -> addition
baz += false // 1

// Number + String -> concatenation
bar += "foo" // "5foo"

// String + Boolean -> concatenation
foo += false // "foofalse"

// String + String -> concatenation
foo += "bar" // "foobar"
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Subtraction_assignment

STRINGS

- Strings are used to represent text

```
1 "Patch my boat with chewing gum"
2 'Monkeys wave goodbye'
```

- (\), *escaping character*- backslash, makes it possible to include characters in a string.

```
1 "This is the first line\nAnd this is the second"
```

```
This is the first line
And this is the second
```

STRINGS

- What you would do if you want the backslash (\) itself to appear in your string?
- “A newline character is written like "\n".” This is how it can be expressed:

```
1 "A newline character is written like \"\\n\"."
```

- **Concatenation?**

```
1 "con" + "cat" + "e" + "nate"
```

```
"concatenate"
```

UNARY OPERATORS

- Not all operators are symbols. Some are written as words.
- `typeof` operator, produces a string value naming the type of the value you give it.

```
1 console.log(typeof 4.5)
2 // → number
3 console.log(typeof "x")
4 // → string
```

- The other operators all operated on two values, but `typeof` takes only one. Operators that use two values are called *binary* operators, while those that take one are called *unary* operators.

JAVASCRIPT COMPARISON AND LOGICAL OPERATORS

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

http://www.w3schools.com/js/js_operators.asp

BOOLEAN VALUES

- Two types:
  - True
  - False

Here is one way to produce Boolean values:

```
1 console.log(3 > 2)
2 // → true
3 console.log(3 < 2)
4 // → false
```

Strings can be compared in the same way.

```
1 console.log("Aardvark" < "Zoroaster")
2 // → true
```

LOGICAL OPERATORS

- Three logical operators: *and*, *or*, and *not*.
- &&: and
- ||: or
- !: not

```
1 console.log(true && false)
2 // → false
3 console.log(true && true)
4 // → true
```

```
1 console.log(false || true)
2 // → true
3 console.log(false || false)
4 // → false
```

UNDEFINED VALUES AND AUTOMATIC TYPE CONVERSION

- There are two special values, written
  - `null` and `undefined`
    - that are used to denote the absence of a meaningful value

- Some odd things:

```
1  console.log(8 * null)
2  // → 0
3  console.log("5" - 1)
4  // → 4
5  console.log("5" + 1)
6  // → 51
7  console.log("five" * 2)
8  // → NaN
9  console.log(false == 0)
10 // → true
```

- How does a program keep an internal state?
- How does it remember things?
- **Variables?**

VARIABLES

- Used to store a value, regardless of type
- A variable must have a name
- May contain letters, numbers, _, and $
- May not start with a number (case-sensitive)

VARIABLES

```
// Variable whose value is a string
var name = "Sebastian";
// Variable whose value is a number
var age = 26;
//Case sensitive
//All these count as different variables
var firstname = "Johannes";
var FirstName = "Johannes";
var FIRSTNAME = "Johannes";
```

VARIABLES (CONT'D)

- Dynamic
- Variables value can be:
  - Changed on request
  - Changed from one type to another
  - Be the result of a calculation,

VARIABLES (CONT'D): EXAMPLE

```
// Initialize the variable "name"
var name = "Sebastian";
// Change the value
name = ”Bato";
//Change the type and value
name = 356;

// The result of a calculation
var width = 10;
var height= 25;
var area = width * height;
```

EXERCISE

Calculate the “age”

1. In a group of three.
2. Declare and store the current year in a variable.
3. Declare and store the birth year in a variable (you can play with 3 sample ages).
4. Declare new variable where you would calculate the age, then output : ”The age is NN", substituting the values.

OPERATOR EXAMPLE

```
// Given these variables:
var name = ”John Doe";
//Change the type and value
var age = 22;

// Is it true or false?
(age > 20 && age < 30) && name != ""
```

TYPES EXAMPLE

```
// A variable without a value:
var name;

// Is of the "undefined”
 typeof name; // undefined

// Comparisons with "false" values
(age > 20 && age < 30) && name != “”
"" == false // true
0 == false // true
```

STRING EXAMPLE

- Selection of Methods
  - length
  - toUpperCase
  - toLowerCase

```
//length, ie, number of characters
“Sherlock”.length; // 8
var name = “Sherlock” ;
name.length; // 8

//Convert to lowercase and uppercase letters
name.toLowerCase; // "sherlock"
name.toUpperCase; // "SHERLOCK"
```

STRING MANIPULATION WITH SPECIAL CHARACTERS

```
var x = 'It\'s alright';
var y = "We are the so-called \"Vikings\" from
the north."
```

The list of special characters that can be added to a text string with the backslash sign

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

http://www.w3schools.com/js/js_strings.asp

DATA STRUCTURES

- An effective way to organize data JavaScript has (among others): `Array` (vector)
  - Consists of a collection of values (elements)
  - Values of various types are allowed

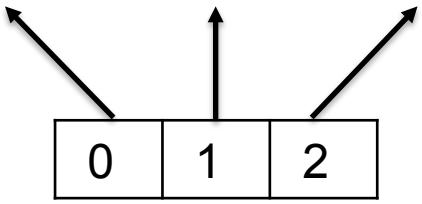
Syntax:

```
var array-name = [item1, item2, ...];
```

DATA STRUCTURES (CONT'D)

- A vector has an "index"
- Each element's position is represented by a number
- First position is 0

```
var names = [ "Sebastian", "Bato", "Nils" ];
```



VECTOR EXAMPLE

```
var names = [ ”Sebastian”, ”Bato”, “Nils” ];


// In order to retrieve a value, we specify a
// Position (index) between [...]


   names [0];
      // ”Sebastian”
   names [2];
      // “Nils”
```

VECTOR EXAMPLE (CONT'D)

```
var names = [”Sebastian”, ”Bato”, “Nils” ];

// A vector is enclosed by [...]
// Each value separated by a comma
var names = [”Soccer”, ”Tennis”, “Golf” ];

// various types of values are allowed:
var person = [”John”, ”Doe”, 28, true];

// Variables can be used as elements
var firstname = ”John”;
var lastname = ”Doe”;
var person = [firstname, lastname];
```

VECTOR EXAMPLE (CONT'D)

```
var names = [”Sebastian”, ”Bato”, “Nils” ];

// Change the value of the element with position 1
names [1] = ”John”;
names; // [”Sebastian”, ”John”, “Nils” ];

// Determine the number of elements in a vector
names.length; // 3
```

VECTOR EXAMPLE (CONT'D)

```
// Create an empty vector
var  names = [];
// Add elements to the end of the vector
names.push (”Sebastian”);
names.push (”Bato”);
names; // [”Sebastian”, ”Bato”];
// Delete the last element
names.pop ();
names; // [”Sebastian”];
// Variables can of course be used
var name = "Sebastian";
names.push(name);
names; // ["Bo", "Sebastian"];
```

VECTOR EXAMPLE (CONT'D)

Strings manipulation - with vector of characters

```
var fullname = "Sherlock Holmes";


fullname[0];  // "S"
fullname[9];  // "H"
```

PROGRAMMING/SCRIPTING CONCEPTS EXPLAINED (VARIABLES, ARRAYS, STRINGS, & LENGTH)

https://www.youtube.com/watch?v=aeoGGabJhAQ

SUMMARY

- Values, Types and operators
  - Numbers
  - Arithmetic operators
  - Assignment operators
  - Strings
  - Boolean values
  - Logical operators
- Variables
- Vectors (data structures)

THANK YOU

QUESTIONS?