JÖNKÖPING UNIVERSITY

School of Engineering

BASICS IN JAVASCRIPT

Web Development with JavaScript and DOM

TWJK14 Spring 2017

Peter Larsson-Green

JÖNKÖPING UNIVERSITY
School of Engineering

INTRODUCTION TO JAVASCRIPT

A programming language browsers interpret.

Created by Brendan Eich 1995 (Netscape).

Has many names:

- LiveScript
- JavaScript
- JScript
- ECMAScript

VERSIONS

JavaScript: 1995 (used in Netscape)

JScript: 1996 (used in IE3)

ECMAScript 1: 1997

ECMAScript 2: 1998 (specification re-written)

ECMAScript 3: 1999

ECMAScript 4: Abandoned.

ECMAScript 5: 2009

ECMAScript 5.1: 2011 (specification re-written)

ECMAScript 6: 2015 ("ECMAScript 2015")

ECMAScript 7: 2016 ("ECMAScript 2016"):

- https://www.ecma-international.org/ecma-262/7.0/index.html

Curios about new features?

- https://github.com/tc39/ecma262/blob/master/README.md

JS IS AN IMPERATIVE LANGUAGE

A program consists of:

- A sequence of statements.

A statement consists of:

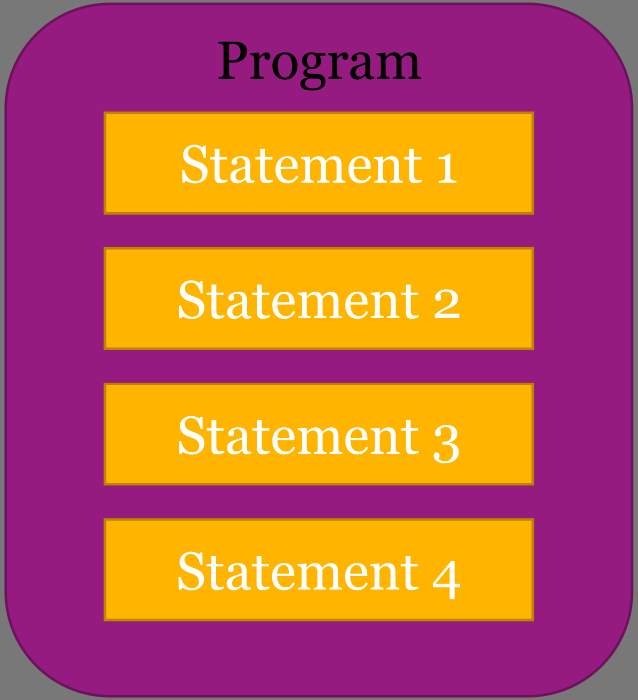
- Other statements and expressions.

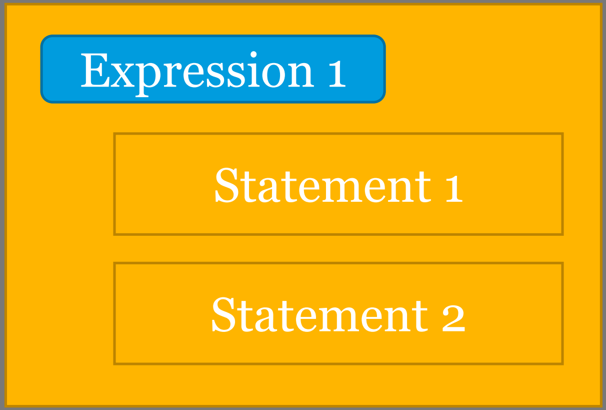
Expressions evaluate to:

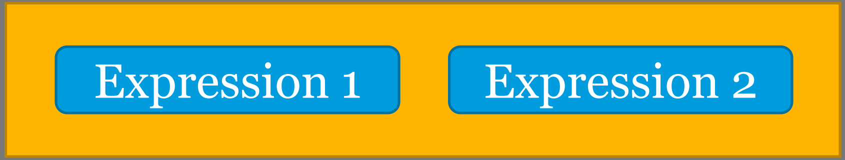
- Values.

Executed statements:

- Alters the state of the program.







Name	Value
x	12
y	36

Variable table.

PROPERTIES OF JAVASCRIPT

- Has dynamic types.
  - The data type is stored in the value, not the variable.

```
var five = 5
five = "5"
```

- Functions are first-class-citizens.
  - Can pass them around as all other values.

PROPERTIES OF JAVASCRIPT

- Has two categories of values:
  - Primitive (Boolean, Number, String, Null, Undefined (and Symbol)).
    - Can be converted into objects automatically.
  - Objects (Boolean, Number, String, Arrays, Functions, ...).
    - A collection of key-value pairs (including methods).
- Objects are prototype based.
  - All objects "inherit" from another object.
  - Objects can be created by a function (which they are instance of).
    - Known as the constructor.

PRIMITIVE VALUES

Are immutable.

Some literal expressions evaluating to primitive values:

- Number: `55`
- Number: `5.5`
- Boolean: `true`
- String: `"Hi!"`
- Null: `null`
- Undefined: `undefined`

NUMBERS

Number objects "inherits" from `Number.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-number-prototype-object

```
var pi = 3.14
var pi_as_string = pi.toString()    // "3.14"
pi_as_string = pi.toFixed(3)        // "3.140"
pi_as_string = pi.toLocaleString() // "3,14"
```

Some special values are stored in global variables:

- `Infinity`
- `NaN` (Not a Number)

```
var pi_as_object = new Number(3.14)
```

NUMBERS

The common mathematical operators are supported:

```
var one = 0 + 1
var two = 4 - 2
var six = 2 * 3
var four = 8 / 2
var eight = 17 % 9
```

```
var number = 1
number += 4 // 5
number -= 2 // 3
number *= 3 // 9
number /= 2 // 4.5
number++ // 5.5
number-- // 4.5
++number // 5.5
--number // 4.5
```

```
Infinity + 5 → Infinity
5 / Infinity → 0
Infinity / Infinity → NaN
NaN + 23     → NaN
```

NUMBERS

The common mathematical operators are supported:

- `1 == 1` → `true`
- `1 != 2` → `true`
- `2 <  1` → `false`
- `2 <= 1` → `false`
- `2 >  1` → `true`
- `2 >= 1` → `true`

BOOLEANS

Boolean objects "inherits" from `Boolean.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-boolean-prototype-object

```
var yes = true
var yes_as_string = yes.toString() // "true"
```

The common logical operators are supported:

```
var no = !true
var yes = true && true
var si = false || true
```

Lazy evaluation!

The `&` and `|` operators exist too!

```
var true_as_object = new Boolean(true)
```

STRINGS

String objects "inherits" from `String.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-string-prototype-object

```
var abc = "abc"
abc = 'abc'
var b = "abc".charAt(1)
var yes = "abc".endsWith('bc')
var one = "abc".indexOf("b")
var adc = "abc".replace("b", "d")
// ...
```

```
var abc_as_object = new String("abc")
```

STRINGS

Comparing strings:

- `"ab" == "ac"` → `false`
- `"ab" != "ac"` → `true`
- `"ab" <  "ac"` → `true`
- `"ab" <= "ac"` → `true`
- `"ab" >  "ac"` → `false`
- `"ab" >= "ac"` → `false`

STRINGS

String operations:

- `"ab" + "ac" → "abac"`
- `"ab" +  3   → "ab3"`
- ` 3   + "ab" → "3ab"`
- `"3"  + "3"  → "33"`
- ` 3   + "3"  → "33"`
- ` 3   - "3"  → 0`
- `"The sum is: " + 1+3 + "." → The sum is: 13.`
- `"The sum is: " +(1+3)+ "." → The sum is: 4.`

OBJECTS

Objects inherits from `Object.prototype` (by default).

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-object-prototype-object

- Store key-value pairs.
  - Keys are casted into strings.

```
var myEmptyObject = {}
```

```
var mySmallObject = {one: 1} // Or: {"one": 1}
var numberOne = mySmallObject.one
var numeroUno = mySmallObject["one"]
mySmallObject.two = 2
mySmallObject["two"] = 2
```

OBJECTS

Objects inherits from `Object.prototype` (by default).

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-object-prototype-object

- Store key-value pairs.
  - Keys are casted into strings.

```
var myLargeObject = {1: "One", 2: "Two", 3: "Three"}
var stringOne = myLargeObject[1]
var stringUno = myLargeObject["1"]
var iAmUndefined = myLargeObject[4]
delete myLargeObject[2]
iAmUndefined = myLargeObject[2]
```

ARRAYS

Array objects inherits from `Array.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-array-prototype-object

- Works more like lists than arrays.
  - Dynamic size.
- Are implemented as objects.

```
var myEmptyArray = []
var mySmallArray = [55]
var myLargeArray = [1, 2, 3, 9, 5, 7]
var six = myLargeArray.length
var nine = myLargeArray[3]
myLargeArray[3] = 4
```

ARRAYS

Array objects inherits from `Array.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-array-prototype-object

- `[1, 2].concat([3, 4])` → `[1, 2, 3, 4]`
- `["a", "b", "c"].indexOf("b")` → `1`
- `[1, 2, 3].join("_")` → `"1_2_3"`

```
var array = [1, 2, 3]
var three = array.pop()
// array = [1, 2]
```

```
var array = [1, 2, 3]
var one = array.shift()
// array = [2, 3]
```

```
var array = [1, 2]
array.push(3)
// array = [1, 2, 3]
```

```
var array = [2, 3]
array.unshift(1)
// array = [1, 2, 3]
```

JÖNKÖPING UNIVERSITY
School of Engineering

VARIABLES

Are created using the `var` statement.

```
var variableName // Initialized to undefined.
```

```
var variableName = value
```

Can be re-assigned using the re-assignment statement:

```
variableName = value
```

FUNCTIONS

- Functions are values (objects).
  - Are stored in variables like ordinary values.
- Create a new scopes (only way before ES6).
- Can access variables outside the function.

Functions without return value returns `undefined`!

```
var numberOfCalls = 0
function average(x, y){
  numberOfCalls++
  var sum = x + y
  return sum / 2
}
var five = average(4, 6)
```

```
var average = function(x, y){

  var sum = x + y
  return sum / 2
}
var five = average(4, 6)
```

IF STATEMENTS

```
function biggest(x, y){
  if(x < y){
    return y
  }else{
    return x
  }
}
var five = biggest(5, 2)
```

```
function sign(n){
  if(n < 0){
    return -1
  }else if(n == 0){
    return 0
  }else{
    return 1
  }
}
var one = sign(99)
```

LOOPS

```
function sum(n){
  var sum = 0
  for(var i=1; i<=n; i++){
    sum += i
  }
  return sum
}
var fifteen = sum(5)
```

```
function sum(n){
  var sum = 0
  while(0 < n){
    sum += n
    n--
  }
  return sum
}
var fifteen = sum(5)
```

LOOPS

```
function sum(n){
  var sum = 0
  do{
    sum += n
    n--
  }while(0 < n)
  return sum
}
var fifteen = sum(5)
```

CONDITIONS

Any value can be used as condition.

- If it is not a boolean value it will be converted:
  - `undefined`, `null`, `NaN`, `0`, and `""` will be converted to `false`.
  - All other values will be converted to `true`.

<u>Examples</u>

`0` is?	Falsey!
`{}` is?	Truthy!
`new Number(0)` is?	Truthy!
`new Boolean(False)` is?	Truthy!

SWITCH STATEMENT

```
function digitToString(d){
  switch(d){
    case 1:
      return "one"
    case 2:
      return "two"
    // ...
  }
}
var two = digitToString(2)
```

```
function getMood(weekday){
  switch(weekday){
    case 1:
    case 3:
      return "Sad"
    case 6:
      return "Happy"
    default:
      return "Angry"
  }
}
var myMood = getMood(4)
```

EXCEPTIONS

```
function compute(operand1, operation, operand2){
  switch(operation){
    case "add":
      return operand1 + operand2
    // ...
    case "div":
      if(operand2 != 0){
        return operand1 / operand2
      }else{
        throw "Division by zero"
      }
  }
}
```

```
try{
  var result = compute(20, "div", 0)
}catch(error){
  if(error == "Division by zero"){
    var result = 9999999999
  }
}finally{
  // Do something with result!
}
```

GLOBAL FUNCTIONS

Some global functions exist.

http://www.ecma-international.org/ecma-262/7.0/#sec-function-properties-of-the-global-object

- `eval("JS code to be executed")`
- `isFinite(123)` → `true`
- `isNaN(123)` → `false`
- `parseFloat("123.45")` → `123.45`
- `parseInt("123", 10)` → `123`
- …

THE GLOBAL OBJECT

All global variables are also added to the global object.

- Is stored in the global variable `window` in browsers.

THE KEYWORD `THIS`

Used in functions to access the caller.

- Will be the global object if called as a function.

```
function globalFunc(){
  // this refers to: the global object.
  function innerFunc(){
    // this refers to: the global object.
  }
  innerFunc()
}
globalFunc()
```

THE KEYWORD `THIS`

Used in functions to access the caller.

- Will be the object calling the function if called as a method.

```
var rectangle = {
  width: 100,
  height: 50,
  getArea: function(){
    return this.width * this.height
  }
}
var area = rectangle.getArea()
var getArea = rectangle.getArea
area = getArea()
```

`this` inside `getArea` will be the global object!

EXPLICITLY SETTING `THIS`

Can be set when calling a function.

- By using the `call` method:
  - `theFunction.call(valueForThis, firstArg, secondArg, ...)`
- By using the `apply` method:
  - `theFunction.apply(valueForThis, [firstArg, secondArg, ...])`

THE `ARGUMENTS` OBJECT

A function can receive more or less arguments than it has parameters.

`arguments` is variable available in functions:

- Contains an object with the arguments passed to the function.
- Looks like an array:
  - Number of arguments stored in the property `length`.
  - Value for the first argument found in the property `0`.
  - Value for the second argument found in the property `1`.
  - ...

EXAMPLE

```
function sumOfArgs(){
  var sum = 0
  for(var i=0; i<arguments.length; i++){
    sum += arguments[i]
  }
  return sum
}
var seven = sumOfArgs(1, 6)
var fourteen = sumOfArgs(3, 5, 6)
```

DEFAULT VALUES FOR PARAMETERS

Parameters with no argument are assigned `undefined`.

Common strategy: Use the `||` operator for default values

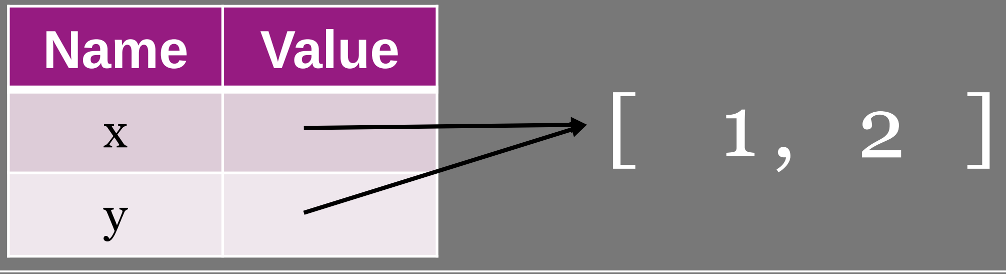
```
function getSum(x, y){
  x = x || 0
  y = y || 0
  return x + y
}
var zero = getSum()
var one = getSum(1)
var two = getSum(1, 1)
```

OBJECTS AND REFERENCES

We never deal directly with objects, only references to them.

- We often create copies of the references.

```
var x = [1]            // A reference to the array is stored in X.
var y = x              // Stores a copy of the reference in Y.
y.push(2)              /* Pushes 2 to the array Y refers to (which
                          is the same as the array X refers to!) */
var two = x.length     // So the array X refers to is affected too!
```



Variable table.

OBJECTS AND REFERENCES

We never deal directly with objects, only references to them.

- We often create copies of the references.
  - E.g. when we pass them to functions.

```
function initialize(rectangle){
  rectangle.width = 100
  rectangle.height = 50
}
var rect = {}
initialize(rect)
var fiveThousand = rect.width * rect.height
```

THE MATH OBJECT

The global variable `Math` stores an object with math values.

http://www.ecma-international.org/ecma-262/7.0/#sec-math-object

- `Math.PI` → `3.14159...`
- `Math.abs(-4)` → `4`
- `Math.ceil(4.5)` → `5`
- `Math.cos(0)` → `1`
- `Math.floor(4.5)` → `4`
- `Math.pow(2, 3)` → `8`
- `Math.random()` → `0.123` (between 0 and 1 (1 excluded))
- `Math.round(4.5)` → `5`

DATES

The function (constructor) `Date` can be used to create date objects.

http://www.ecma-international.org/ecma-262/7.0/#sec-date-constructor

```
var today = new Date()
var christmas = new Date(2016, 11, 24, 15, 0, 0, 0)
var unixEpochStart = new Date(0)
var unixEpochStartNextDay = new Date(24*60*60*1000)
```

DATES

Date objects "inherits" from `Date.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-date-prototype-object

```
var today = new Date() // 2016-05-04 08:51:43.398 (Wednesday)
var year = today.getFullYear() // 2016
var month = today.getMonth() // 4
var date = today.getDate() // 4
var hours = date.getHours() // 8
var minutes = date.getMinutes() // 51
var seconds = date.getSeconds() // 43
var milliseconds = date.getMilliseconds() // 398
var weekDay = date.getDay() // 3
```

DATES

Date objects "inherits" from `Date.prototype`.

http://www.ecma-international.org/ecma-262/7.0/#sec-properties-of-the-date-prototype-object

```
var today = new Date() // 2016-05-04 08:51:43.398 (Wednesday)
var year = today.getFullYear() // 2016
// ...
```

For each `get*` method, there is also `set*` method.

For each `get*` method, there is also a `getUTC*` method.

For each `getUTC*` method, there is also a `setUTC*` method.

```
var millisecondsSinceEpochStart = theDate.valueOf()
```

A COMMON MISTAKE

```
var funcs = []
for(var i=0; i<5; i++){
  funcs.push(function(){ return i })
}
var sum = 0
for(var j=0; j<5; j++){
  sum += funcs[j]()
}
// sum = 25?!
```

AVOIDING GLOBAL VARIABLES

Global variables make it hard to mix code from different sources.

- E.g., different libraries might create same global variable ☹

Functions are the only way to creates new scopes.

- Create an anonymous function and call it directly ☺

```
var iAmAGlobalVariable = "☹"
```

```
(function(){
  var iAmALocalVariable = "☺"
})()
```

STRICT MODE

JavaScript does stupid things to keep your code from crashing.

- Obvious errors remain hidden ☹

Strict mode throws exceptions instead.

- Added in ECMAScript 5.
- To activate it:

```
"use strict"
// Your strict JavaScript code here.
```

```
function aStrictFunction(){
  "use strict"
  // Your strict JavaScript code here.
}
```

STRICT EXAMPLES

Re-assignment to none-existing variable:

```
x = 12
```

```
function aStrictFunction(){
  x = 12
}
```

None strict

Create it as a global variable.

Strict

Throw an exception.

STRICT EXAMPLES

Using same property in object:

```
var object = {x: 1, x: 2}
```

None strict

Use the last value.

Strict

Throw an exception.

STRICT EXAMPLES

Using same parameter in functions:

```
var myFun(parameter1, parameter1){ }
```

None strict

Use the last argument.

Strict

Throw an exception.

STRICT EXAMPLES

Using `this` in ordinary functions:

```
function doSomething(){
  return this.x * this.y
}
doSomething()
```

None strict

Let `this` refer to the global object.

Strict

Let this refer to `undefined`.

STRICT EXAMPLES

The function statement inside if statements.

```
if(true){
  function test(){
  }
}
```

None strict

Make `test` to a global function.

Strict

Make `test` to a local function in the if statement.

COMPARING VALUES

JavaScript automatically converts operands.

```
1 == 1 →                   true
1 == new Number(1) →       true
{} == {} →                 false
[] == [] →                 false
var a = []; a == a →       true
[1] == "1" →               true
[1, 2] == "1,2" →          true
new Number(1) == new Number(1) →  false
```

Use === instead of == and !== instead of != if you don't want JavaScript to automatically convert the operands to same data type!

WHERE TO WRITE JS CODE

1. In event attributes:

```
<p onclick="JavaScript-CODE">Some text</p>
```

   - Can't re-use our JavaScript code on other elements ☹
   - Shouldn't mix HTML and JavaScript code ☹

2. In the `<script>` element:

```
<script type="text/javascript">JavaScript-CODE</script>
```

   - Can't re-use our JavaScript code in other files ☹

WHERE TO WRITE JS CODE

3. In a separate `.js` file:

```
<script src="the-js-file.js"></script>
```

```
JavaScript-CODE
```

- Can use the same JavaScript code in multiple files ☺
- JavaScript files can be cached ☺

DISPLAYING VALUES

JavaScript does not have any print function.

- Values are usually shown by manipulating the DOM.
  - Not an efficient way for debugging and testing.
- The browser gives you the `alert` function:
  - `alert("This string is shown to you!")`
- Most browsers gives you the `console` object:
  - `console.log("This string is shown in the console!")`
  - https://developer.mozilla.org/en-US/docs/Web/API/Console

RECOMMENDED WATCHING

Douglas Crockford:

- The JavaScript Programming Language (2007):
  - https://www.youtube.com/watch?v=v2ifWcnQs6M
- The State and Future of JavaScript (2009):
  - https://www.youtube.com/watch?v=V1_Y-KVhZ9Q
- Videos on YouTube:
  - https://www.youtube.com/playlist?list=PLEzQf147-uEpvTa1bHDNlxUL2klHUMHJu

RECOMMENDED READING

You Don't Know JS:

- Up and Going:
  - https://github.com/getify/You-Dont-Know-JS/tree/master/up%20%26%20going
- Types & Grammar:
  - https://github.com/getify/You-Dont-Know-JS/blob/master/types%20&%20grammar/README.md
- Scope & Closures:
  - https://github.com/getify/You-Dont-Know-JS/blob/master/scope%20&%20closures/README.md

RECOMMENDED READING

W3Schools:

- JavaScript Tutorial:
  - https://www.w3schools.com/js/default.asp
  - Note: Do not distinguish JavaScript from DOM & BOM.

ECMAScript 7.0 Specification

- http://www.ecma-international.org/ecma-262/7.0