

# JAVASCRIPT PITFALLS



# Agenda

2

- Understand the major differences between popular static languages (C++/C#/Java) and JavaScript
- Variables
- Arrays
- Functions
- Objects
- Modules

# JavaScript is dynamic

3

- You don't specify the data type of a variable when you declare it
- The same variable can point to different data types
- We use **var** to declare a variable
- A variable has a scope
  - ▣ Global variables should be avoided (like in any other object oriented language)

```
var answer = 42;  
answer = "Meaning of life";
```

# Declaring Variables

4

- Case sensitive
- \$ and \_ are valid variable names
  - ▣ And common
- Cannot use reserved keywords
- Usually, camel case convention

```
$(function () {  
    var res = _.map([1, 2, 3], function (num) {  
        return num * 2;  
    });  
});
```

- Do you like above code ?

# Implicit Variable Declaration

5

- ❑ You can write into a variable even when this variable was not declared before
- ❑ Don't do this !
- ❑ In this case a global variable is created

```
function () {  
    global = 12;  
    var local = "abc";  
}  
  
alert(local);
```

# Automatic Initialization

6

- Like other modern programming languages, JavaScript supports automatic initialization
- The value of uninitialized variable is **undefined**
  - ▣ Not the same as **null** value

```
var num;  
  
console.log(num == undefined);
```

# Undeclared Variable

7

- You cannot read a value of undeclared variable

```
try {  
    if (xxx == 10) {  
    }  
}  
catch (e) {  
    console.log(e.message);  
}
```

- You can ask for the **typeof** of an undeclared variable

```
console.log(typeof xxx);
```



“undefined”

# Window is the Global Scope

8

- Every global variable is a property of a global object named **window**

```
var num = 10;  
console.log(window.num); //prints 10  
  
window.num = 11;  
console.log(num); // prints 11
```

- Objects in JavaScript are dynamic → Global scope is dynamic 😊
  - ▣ See next slides about objects



# Built-in types

9

- JavaScript supports only the following types:
  - ▣ number
  - ▣ boolean
  - ▣ string
  - ▣ function
  - ▣ object
  - ▣ undefined
- Given a variable you can use the keyword **typeof** to read it's runtime type

# Built-in types

10

```
console.log(typeof 1); // number
console.log(typeof 1.2); // number
console.log(typeof "abc"); // string
console.log(typeof "abc"[0]); // string
console.log(typeof true); // boolean
console.log(typeof function () { }); // function
console.log(typeof {}); // object
console.log(typeof null); // object
console.log(typeof new Date()); // object
console.log(typeof window); // object
console.log(typeof undefined); // undefined
console.log(typeof blabla); // undefined
```

# Value vs. Reference type

11

- Same concept as in Java/C#
- Built-in data types are grouped into
  - ▣ Reference types (object, array and function)
  - ▣ Value types (others ...)
- A reference is implemented as a pointer
  - ▣ Points to an object that resides inside the heap
  - ▣ Many references can point to the same object
- A value can only copied
  - ▣ You cannot get the address of value

# Number

12

- There is no distinction between integer and double
- All type of numbers are represented as 64bit floating point values
  - ▣  $10/3 = 3.3333$  not 3
- **parseInt** can be used to parse a string into a number. In case of failure **NaN** is returned

```
var str = document.getElementById("firstName").value;
if (isNaN(parseInt(str))) {
    alert("Please enter a number");
}
```

# String (1)

13

- String contains any Unicode character
- No character type
  - ▣ `str[0]` is also a string !!!
- String literal can be expressed using “ or ‘

```
var str = "ABC";  
var str = 'ABC';
```

- Strings are immutable
  - ▣ Allows for runtime optimization

```
var str = "ABC";  
str[0] = "X";
```



str is still  
“ABC”

# String (2)

14

- Should we use “ or ‘ when writing string literals?
  - ▣ Probably a matter of style
  - ▣ Programmers with C++\Java\C# background tend to use double quotes
  - ▣ Veteran Web Programmers tend to use single quote
- You should be aware of the following
  - ▣ JSON requires double quotes
  - ▣ HTML/XML attributes are usually expressed using double quotes
    - Therefore, when building XML fragments at runtime it is easier to use single quote for the whole string literal

# String's useful methods

15

- **charAt** – Returns the character at the specified index
- **charCodeAt** – Returns the Unicode value
- **indexOf** – Returns the position
- **match** – Matching a regular expression
- **trim** – Removes whitespaces from both sides
- **split** – Splits a string into an array of substrings
- More ...

# Undefined

16

- A special data type
- Has only one value named **undefined**
- The value **undefined** is important concept in JavaScript
- You may encounter it during several scenarios
  - ▣ Uninitialized variable
  - ▣ A function without a return value
  - ▣ A function parameter that was not specified by the caller
  - ▣ A non existent object property
  - ▣ A non initialized array index



# Comparison Operators

17

- JavaScript has both **strict** and **abstract** comparisons
- A strict comparison is only true if the operands are the same type
- Abstract comparison converts the operands to the same type before making the comparison

```
console.log(0 == false);  
console.log(2 == "2");  
console.log(undefined == null);
```

```
console.log(0 === false);  
console.log(2 === "2");  
console.log(undefined === null);
```

# Data Type Conversion

18

- Data types are converted automatically as needed during script execution
- Operator `+` may convert numeric values to strings

```
var num = 10;  
alert(num + "0");
```



100

- Other operators may convert string values to numeric

```
var num = 10;  
alert(num * "2");
```



20

# Conversion Tricks

19

- Some JavaScript programmers use operators `+` and `*` to convert data types
- Convert string to number

```
var str = document.getElementById("firstName").value;  
if (isNaN(str * 1)) {  
    alert("Please enter a number");  
}
```

- Convert number to string

```
var num = 10;  
console.log(num + "");
```

# Logical Operators

20

- Typically used with Boolean values
  - ▣ In that case, they return a Boolean value
  - ▣ Behavior is consistent with other static programming languages (C++/Java/C#)
- May be used with non Boolean values
  - ▣ In that case, they return a non-Boolean value

```
alert("dog" || "cat")
```



"dog"

```
alert("dog" && "cat")
```



"cat"

# Array

21

- Array is created using the following syntax
  - []
  - new Array

Preferred

```
var arr = [];  
var arr = [1,2,3];
```

Less common

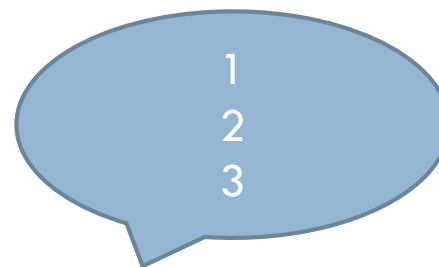
```
var arr = new Array();  
var arr = new Array(10); // length is 10  
var arr = new Array(10, 2); // length is 2
```

# Iterating an Array

22

- Straight forward
- Use a running index and the **length** property

```
var arr = [1, 2, 3];  
  
for (var i = 0; i < arr.length; i++) {  
    console.log(arr[i]);  
}
```



# Array is dynamic

23

- New elements can be added/deleted at runtime
  - ▣ In contrast to static languages
- The property **length** is automatically being updated

```
var arr = [];  
arr.push(10); // add last  
arr.pop(); // remove last  
arr.splice(arr.length-1, 1); // remove last  
arr[10] = 10; // never throws an exception  
arr.length = 2; // resize  
arr.shift(); // remove first  
arr.splice(0, 1); // remove first
```

# Useful Array's Methods

24

- **concat** - Joins two or more arrays, and returns a copy of the joined arrays
- **indexOf** - Search the array for an element and returns its position
- **join** - Joins all elements of an array into a string
- **sort** - Sorts the elements of an array
- **toString** - Converts an array to a string, and returns the result



# Object

25

- A container of keys and values
- The key must be of type string
- Has built-in methods
- Creating empty object is easy

```
var obj = {};  
alert(typeof obj);
```



“object”

Less common

```
var obj = new Object();  
alert(typeof obj);
```



“object”

# Initializing an Object

26

- An object can be initialized at declaration
- A.K.A object literal syntax (the basis for JSON)

```
var obj = {  
  id: 123,  
  name: "Udi",  
  email: "udi@gmail.com"  
};
```

Less common

```
var obj = {  
  "id": 123,  
  "name": "Udi",  
  "email": "udi@gmail.com"  
};
```

# Object is dynamic

27

- Properties can be added/removed after creation

```
var obj = {};  
obj.name = "Ori";  
obj["name"] = "Ori";
```

- Removing a property

```
delete obj["name"];  
delete obj.name;
```

- Accessing non existent property yields the value **undefined**

# Object Content

28

- The `for...in` statement allows you to iterate over all object's properties

```
var obj = {  
  "id": 123,  
  "name": "Roni",  
  "email": "roni@gmail.com"  
};  
  
for (var key in obj) {  
  var value = obj[key];  
  console.log(key + " = " + value);  
}
```



```
id = 123  
name = roni  
email = roni@gmail.com
```

# Object's built-in methods

29

- **hasOwnProperty** - Returns a Boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain
- **toString** - Returns a string representation of the object
- **valueOf** - Returns the primitive value of the specified object
- More ...

# Array is an Object

30

- ❑ You can act on a array is if it was an object (it is !)
- ❑ Not recommended
- ❑ What is the expected output?

```
var arr = [1, 2, 3];  
arr.name = "Ori";  
  
for (var i = 0; i < arr.length; i++) {  
    console.log(arr[i]);  
}  
  
for (var key in arr) {  
    console.log(arr[key]);  
}
```

# Function

31

- More than just a method ...
  - ▣ The basic for advanced JavaScript techniques
- Declaring a function
- Calling a function is also straightforward

```
function add(num1, num2) {  
    return num1 + num2;  
}
```

```
var res = add(num1, num2);
```

# Pass by value

32

- JavaScript only supports “pass by value” mechanism
- The parameter being sent to a function is copied
  - ▣ Whether it is a reference or a value

```
var str = "ABC";  
  
function modify(str) {  
    str = "XXX";  
}  
  
console.log(str);
```



# Where to declare variables ?

33

- A variable is accessibly inside its surrounding function
- Even before point of declaration
- Therefore many JavaScript programmers declare all variables at the beginning of the method

```
var num = 11;  
  
function doSomething() {  
    console.log(num);  
    var num = 10;  
}  
  
doSomething();
```

# Overloading

34

- ❑ JavaScript does not support Overloading
- ❑ Last method wins
- ❑ You can simulate it

```
var ERR = "ERR";  
var WRN = "WRN";  
var MSG = "MSG";  
  
function log(type, message) {  
    if (message == undefined) {  
        message = type;  
        type = MSG;  
    }  
  
    console.log(type + " " + message);  
}
```

```
log(ERR, "Internal Error");  
log("Connecting to server");
```

# Function – The Dark Side

35

- A function is an object

```
function f() {  
    var num = 10;  
}  
  
f.num = 10;
```

- Has built-in properties and methods

```
function f(input) {  
    console.log(f.name); // the name of the method  
    console.log(f.length); // number of parameters  
    console.log(f.toString()); //function source code  
    console.log(f.arguments); // available only during execution  
    console.log(f.caller.name); // available only during execution  
}
```

# Function – Indirect Invocation

36

- A function can be invoked using special syntax

```
function f(name) {  
    console.log("Hello " + name);  
}  
  
f.call({}, "Ori");  
f.apply({}, ["Ori"]);
```

- Although not intuitive, above syntax is quite common
- Mainly, when doing Object Oriented JavaScript

# Function creates a Scope

37

- Function creates a new scope which is isolated from outer scope
- Outer scope cannot access local variables of a function

```
var num = 20;

function f() {
  var num = 10;

  console.log(num); // yields 10
}

f();

console.log(f.num); // yields undefined
```

# Closure

38

- Inner function may access the local variables of the outer function
  - ▣ Even after outer function completes execution
- Allows us to simulate stateful function

```
function getCounter() {  
  var num = 0;  
  function f() {  
    ++num;  
    console.log("Num is " + num);  
  }  
  return f;  
}
```

```
var counter = getCounter();  
counter();  
counter();
```

# Function inside an Object

39

- An object can contain functions

```
var obj = {  
  dump: function() {  
    console.log("dumping...");  
  }  
};  
  
obj.dump();
```

- Feels like OOP
- The keyword **this** is used for accessing other properties (see next slide)

# The this keyword

40

- Available only inside a function
- Points to the object that this function is being invoked on

```
var obj = {  
  id: 123,  
  dump: function() {  
    console.log(this.id);  
  }  
};  
  
obj.dump();
```

- Global function points to the window object



# Apply & Call - Recap

41

- You can control the value of **this** using **apply** and **call** methods

```
var obj = {  
  id: 123  
};  
  
function dump() {  
  console.log(this.id);  
}  
  
dump.call(obj);
```

# Self Executing Function

42

- A function can declared without a name
- Since no name exist no one can invoked it
- Except the code that declared it
- A.K.A self executing function

```
(function () {  
    // External code has no access to these variables  
    var url = "http://www.google.com";  
    var productKey = "ABC";  
})();
```

# Sending Parameters

43

- Think about the \$ sign
- Usually it points to jQuery global object
- But how can we ensure that?
  - ▣ There might be a case where additional 3<sup>rd</sup> party library overrides it

```
(function ($) {  
    $.ajax({  
        url: "www.google.com",  
        type: "GET",  
    });  
})(jQuery);
```

# Module

44

- Arrange your JavaScript code into modules
- Each module is surrounded with self executing function thus hiding all local variables and functions
- Peek the ones that should be public (sparsely)

```
var Server = (function () {  
    var baseUrl = "http://www.google.com";  
  
    function httpGet(relativeUrl) {  
        $.ajax(...);  
    }  
  
    return {  
        httpGet: httpGet,  
    };  
})();
```

# Summary

45

- JavaScript is simple but powerful
  - ▣ Small amount of built-in types
  - ▣ Implicit conversion
  - ▣ No character data type
  - ▣ No integer data type
- Function is the basis for advanced JavaScript coding
- Arrange your code into modules