Hello

Javascript

coding_
academy

Introduction

JavaScript (JS) is an interpreted computer programming language.

As part of web browsers, implementations allow client-side scripts to:

- interact with the user
- control the browser
- communicate asynchronously

Introduction

- Javascript is now used to create single page web applications, mobile applications, client side and server side (using node.js).
- Javascript is becoming a true end to end choice for companies and secured its place in the pages of technology-history.

Introduction

- There's no connection to Java…
- Developed by the ECMA organization.
  (European Computer Manufacturers Association)
- Brendan Eich created the first version back in 1995 in ten days (!)

The basics

- Here is a simple script:

```
<body>
<script>
   alert('Hello Javascript!');
</script>
</body>
```

- You can have as many <script> tags as you like
- Usually you put all javascript at the bottom of the page (just above the </body>)
- Note: Javascript is case sensitive.

Comments

Comments are important for any programming language:

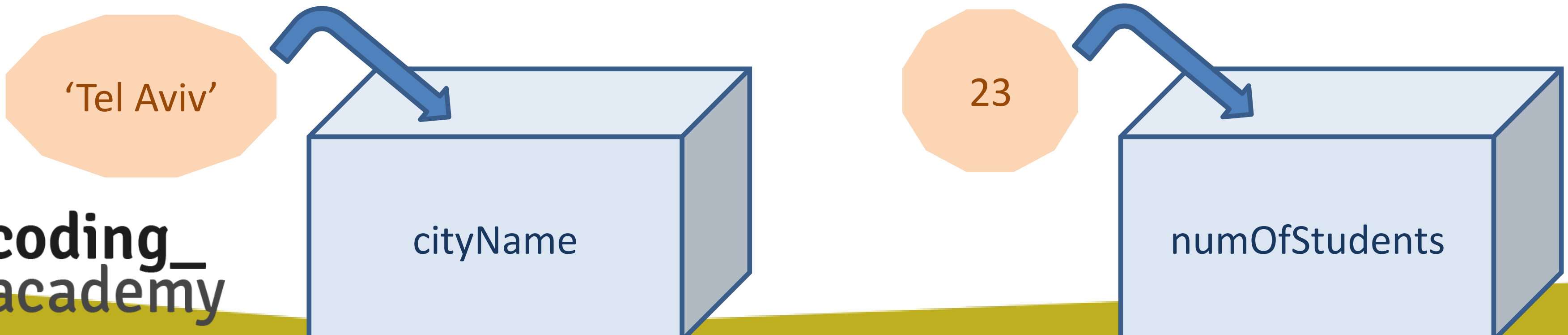
- Make your code more **readable**
- Helps **debugging** your code by canceling parts of it

```
<script>

// I am a comment, and proud of it!

/*
   This is a multi-line
   comment.
*/

// alert('Hello');

</script>
```

Variables

A variable is used to store a value

- it has a name and a value
- Variable names are case-sensitive
- variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores



Variables

Always define variables you use
(using the *var* keyword)

```
var age = 18;
var name = ‘Puki’;
var isHappy = true;

alert(name);
```

Data types

The primitive data types in JavaScript are:

- number   7
- string   ‘Hello There’
- boolean (true or false),
- null
- undefined

Binary Numeric Operators

- Assume we have 2 variables:

  a = 7 b =5

Name	Example	v	Explanation
Negation	v = -a	-7	Opposite of a.
Addition	v = a + b	12	Sum of a and b.
Subtraction	v = a - b	2	Difference of a and b.
Multiplication	v = a * b	35	Product of a and b.
Division	v = a / b	1.4	Quotient of a and b.
Modulus	v = a % b	2	Remainder of a divided by b.

String

- String is a sequence of characters
- You can concatenate strings together with the operator +

```
greet = ‘Hello ’ + name
```

Reading input from user

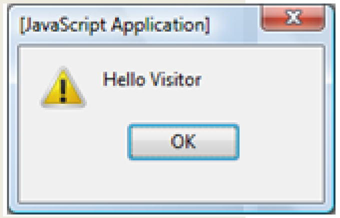
- Every program gets some input from somewhere, does some processing and deliver some output.
- Later on, when we combine JS with HTML we will have better ways to get input from users.
- For now, this is how you read a string from the user in JS:

```
var name = prompt(‘Enter your name’);
```

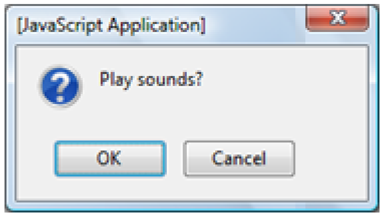
Built in popups

- There are 3 built-in popup windows that can be opened in JS:
  - Alert – shows a msg and an OK button.
  - Confirm – shows a msg, and Ok/Cancel buttons.
  - Prompt – shows a msg and gets input from the user.

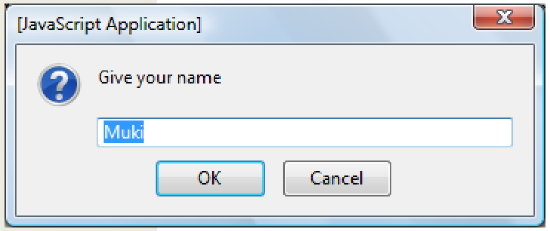
```
alert("Hello Visitor");
var isPlayingSounds = confirm("Play sounds?");
if (isPlayingSounds) {
// play sounds
}
var userName = prompt("Give your name");
alert(userName + " Welcome!");
```


[JavaScript Application]

Hello Visitor


[JavaScript Application]

Play sounds?


[JavaScript Application]

Give your name

Muki

Hands-on

- Read the name of the user and greet him (Hello Muki)
- Get 2 numbers, and print to the console the result of all the operators: +, -, *, /, %

coding_ academy

Unary Operators

Operators that work on a single operand:

Name	Example	Result
Pre-increment	++a	Increments a by one, then returns a.
Post-increment	a++	Returns a, then increments a by one.
Pre-decrement	--a	Decrements a by one, then returns a.
Post-decrement	a--	Returns a, then decrements a by one.

coding_
academy

Unary Operators

Here is a sample:

```
var a = 5;
console.log( "Should be 5: " , a++ );
console.log( "Should be 6: " , a );

var b = 5;
console.log( "Should be 6: " , ++b);
console.log( "Should be 6: " , b )
```

coding_
academy

Booleans and Conditions

Boolean expressions have only two possible values: *true* or *false*.

- e.g. here is a boolean expression: *num > 7*
- This is the building block for **conditions**

```
if (grade >= 70) {
     alert( “Passed” );
} else {
     alert(“Failed” );
}
```

Boolean operations

Name	Example	Result
Equal	a == b	TRUE if a is equal to b. DON'T USE
Identical	a === b	TRUE if a is equal to b, and they are of the same type.
Not equal	a != b	TRUE if a is not equal to b. DON'T USE
Not identical	a !== b	TRUE if a is not equal to b, or they are not of the same type
Less than	a < b	TRUE if a is strictly less than b.
Greater than	a > b	TRUE if a is strictly greater than b.
Less than or equal to	a <= b	TRUE if a is less than or equal to b.
Greater than or equal to	a >= b	TRUE if a is greater than or equal to b.

{coding_
academy

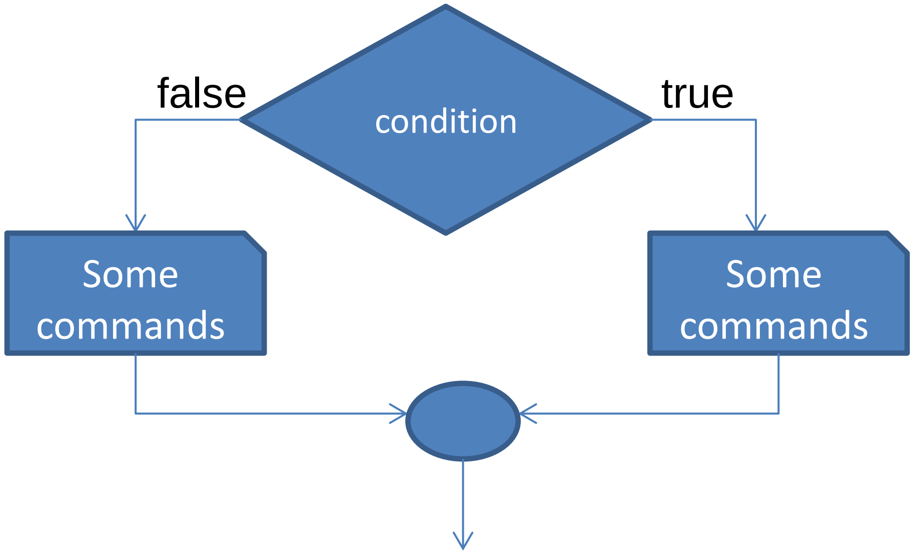
Boolean operations

Name	Example	Result
Not	! a	TRUE if a is not TRUE.
And	a && b	TRUE if both a and b are TRUE.
Or	a || b	TRUE if either a or b is TRUE.

Here *a* and *b* are boolean values (*true* or *false*) or boolean expressions such as *x>7*

Conditions

- Conditions are the most basic flow-control
  - It is called flow-control as it control the flow of our program
  - Based on a condition, our program will choose to execute certain commands and not others.
- Try to have the following graphic image in your mind:



Else If

- Here is a an example:

```
if (grade > 85) {
    alert( “Great!” );
} else if (grade > 70) {
       alert (“OK…”);
} else {
       alert (“Not Good…”);
}
```

Else If

- Here is another example:
  - When will each output be printed?

```
var userIsActivated = false;

if (itemCount < 100) {
   alert("Contact the Supplier");
} else if (userIsActivated){
   alert("You may order");
} else {
   alert("Please activate your Account");
}
```

Live Coding - Conditions

- Read 2 numbers and print the bigger one
- Read 2 numbers and check if either of them is a divider of the other
- Read 3 numbers and check if they could be valid triangle sides, and which triangle are they
- Read 3 grades, check that they are in range 0-100, if so, print their average.

Assignment operators

Name	Example	Result
Assignment	a = b	Stores b value in a
Assign-plus	a += b	a = a + b;
Assign-minus	a -= b	a = a - b;
Assign-multiply	a *= b	a = a * b;
Assign-divide	a /= b	a = a / b;
Assign-concat	str += name	str = str + name

Operators ordering

- Arithmetic before Boolean operators
- AND before OR
- You can always use () to make sure and to increase readability

Short-Circuit

- (Like most programming languages) Javascript evaluates boolean expressions in a short-circuit manner, expression is checked from left to right:
  If the first condition is falsy, the second condition
  is not evaluated.
    - If (found && goLookAgain())
  - If the first condition is truthy, the second condition is not evaluated.
    - If (found || goLookAgain())
- So Javascript stops evaluating a logical expression as soon as the result is determined.

What’s true

- False is
  - The value false
  - The value null
  - The value 0
  - The value undefined
  - The empty string “”
  - And the number NaN (Yes, NaN is a special number…).
- The rest are true

Strict mode

- It is also advised to use the strict mode when developing javascript.
  - Syntax is a bit funny, but it will make you write better code, as a beginning it will not allow use of uninitialized variables.
  - Put it at the beginning of your javascript code

```
'use strict';
```

Loops

- Loops enable us to do something repeatedly
- This is a very important flow control
- *While* is the most generic loop structure:

```
while  (condition ) {
   // do something while condition is true
}
```

While Loop

- What does this code do?
- Lets perform a dry-run on this.

```
var num = 98765;
while(num > 0) {
   var digit = num % 10;
   console.log (digit);
   num = parseInt(num/10);
}
```

While Loop

Sometimes, the loop is dependent on input with some signal for end-of-input, in those cases its best to use the following logic:

```
var choice = +prompt('Please enter your choice (0 to exit):‘);

while (choice !== 0) {
   alert( "Your choice is " + choice );
   // TODO: handle the user choice here
   choice = +prompt('Please enter your choice (0 to exit):‘);
}
alert ("Bye“);
```

Loops – hands-on

- Read positive numbers until their sum is bigger than 100 then print the average
- Read numbers until you get an odd number, then print the maximal even number you got
- Read a number and check if that number is prime
- Read names until “QUIT” is entered then print the names separated by *
- Read 10 numbers and print: sum, max, min
- Read a 5 digits number and check if it symmetric

For Loop

When you know how many times the loop will iterate, the For loop is more convenient.

E.g.: Read 10 numbers and print their sum:

```
var sum = 0;
for (var i=0; i<10; i++) {
    var num = prompt( 'Enter num:’);
    sum += num;
}
alert( ‘sum: ‘ + sum);
```

Loops – hands-on

- Read 3 numbers and print: average, max, min (using a for loop)
- Read the number of kids of the user, then ask him the age for each kid, then print the youngest age.
- Print the multiplication table
- Measure how much time it takes to sum *many* random numbers.

Breaking out from a loop

To exit a loop, use break;

(it works on any kind of loop)

```
for (var i=1; i<=4; i++) {
    if (i % 3 === 0) break;
    console.log('Iteration Number: ' + i );
}
alert( 'done');
```

```
Iteration Number: 1
Iteration Number: 2
```

Also, note how a single command in an if does not need {}

coding_
academy

Continue to next iteration

- To skip the current iteration in the loop, use continue;

```
for (var i=1; i <= 4; i++) {
     if (i % 3 === 0) continue;
     console.log( 'Iteration Number: ' + i);
}
alert ('done');
```

```
Iteration Number: 1
Iteration Number: 2
Iteration Number: 4
```

coding_
academy

Another example

```
// 10 times, but print only odd numbers:
for (var i=1;i<=10;i++){
    if (i % 2 == 0) continue;
    console.log(i + ", ");
}

// Untill you find, unless you get tired:
while (!found) {
    found = lookSomewhere();
    if (tiredOfLooking()) break;
}
```

Functions

- Functions are a way to pack some functionality in a reusable way.
  - A function is a portion of code which performs a specific task.
- It is also relatively independent of the other code.
- Write a function if:
  - You find yourself writing the same piece of code over and over again.
  - You find yourself "copying & pasting" code so that you can re-use it in another part of your application.

Functions

- The problem with copy & paste is that you create more code to maintain.
- If you want to change the code, you need to change it in many different places.
- Here is a simple function:

```
function sayHello() {
   alert (‘Hello’);
}

sayHello();
```

Functions

- A function often accepts one or more "parameters" which are passed to it from outside as “arguments”
- By providing a different argument to a function you can get different results.

```
function sayHello(name) {
   alert( "Hello " + name );
}

sayHello("Muki");
sayHello("Puki");
```

Functions

- Many times, we would like the function to return some calculated value as output.
- Here is a simple example:

```
function calcuateSum(num1, num2) {
   var sum = num1 + num2;
   return sum;
}

var theSum = calcuateSum(5, 9);
alert( "Sum of 5 and 9 is: " + theSum );
```

coding_
academy

Functions

```
function calcuateSum(num1, num2) {
    var sum = num1 + num2;
    return sum;
}

var theSum = calcuateSum(5, 9);
alert( "Sum of 5 and 9 is: " + theSum );
```

- num1 and num2 are **parameters**, being passed the **values** of 5 and 9 **respectably** (as arguments).
- sum is a **local variable**, which **live** and **known** only within the **scope** of the function.
- theSum is a variable in the **Global scope**, its get assigned to the **return value** of the function.

Functions – Hands-on

- Some program need to validate that numbers are ni specific range and are even.
  - write a function isValid(num, minRange, maxRange) that returns true if num is an even number in the given range.
- Write a function that gets 3 sides of a triangle and tells the user if it is:
  - not a triangle
  - all-sides-equals triangle
  - 2-sides-equal triangle
  - Pythagoras triangle

coding_ academy

Functions – Hands-on

- Write a function: printPrimes that gets 2 numbers: minRange and maxRange and prints all the prime numbers in that range.

Functions

- Functions can define parameters
- Functions can return a value
- Functions can define local variables
- Parameters are generally passed by-value, it means that the parameter gets a copy of the sent argument.

- Functions can use global variables
  (we should limit the usage of global variables)

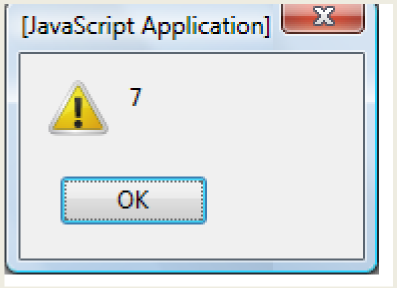
Variables’ scopes

- Local variables
  - Declared inside scopes (such as a function)
  - Live from declaration to the end of the scope
- Global variables
  - Declared outside any function
  - Can be used throughout the page and inside functions
  - Live from declaration

Variables' scopes

- Important JS fact:
  - Variables are not only declared implicitly when you set a value, they can be also made global. Have a look at the following code:

```
function initSelf() {
   max = 7;
   gigi();
}

function gigi() {
   alert(max);
}
```



  - Always define your variables (with var)
  - And *use strict* to watch your back

Short If

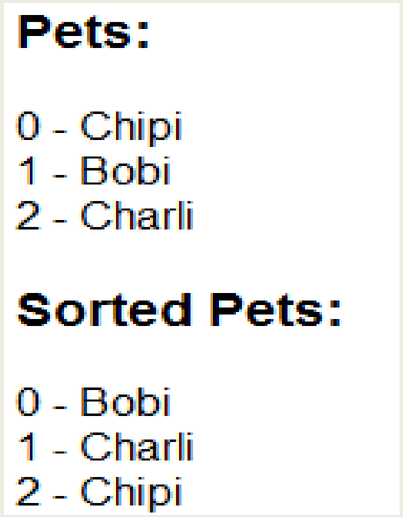
When your if statement is just setting 2 values,
(one if true and another if false) - prefer the shortened syntax:

```
var a = 5;
var b = 10;

var result = (a > b)? a : b;
console.log ( "The result is: " + result );
```

Arrays

It is often needed to store multiple values in a single variable

```
// Don't use the Array syntax
// var pets = new Array();
var pets = [];

pets[0] = "Chipi";
pets[1] = "Bobi";
pets[2] = "Charli";

pets.sort();
```



```
Pets:

0 - Chipi
1 - Bobi
2 - Charli

Sorted Pets:

0 - Bobi
1 - Charli
2 - Chipi
```

Looping through an array

Examples:

```
var scores = [67, 73, 82, 48];
var total = 0;
for (var i = 0; i < scores.length; i++) {
   total += scores[i];
}
var avg = total / scores.length; // 67.5
```

References vs Values

The array name is a reference to the beginning of the array. Consider the following code:

```
var num = 12;
var num1 = num;
num1 = 7

var nums = [67, 73];
var nums1 = nums;
nums.push(7)
nums1 = []
```

References vs Values

Same idea applies when passing an array as argument to a function:
Consider the following code:

```
function foo(nums) {
  nums.push(7)
  nums = []
}
```

Arrays – hands-on

Write a program that generate 10 random numbers, sort them and print them from the biggest to the smallest

switch

- To easily code several conditions based on a single value, use the switch command:

```
switch (itemCode) {
case 101:
   // handle code: 101 - Dog
   break;
case 102:
   // handle code: 102 - Cat
   break;
default:
   // when no case covered
}
```

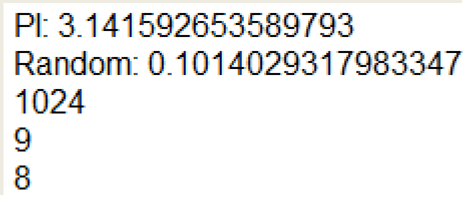
Built-in Classes and Objects

- Javascript supplies some built-in classes for your use:
  - String, Date, Array, Math, etc.
- When running Javascript in a browser, we also have access to some objects:
  - window, document, location, etc.
- Lets examine some of them closely.

Math

- Use the Math (static) methods to do mathematical calculations:
  - max receives an array of numbers.
  - random returns a real number between 0-1.

```
console.log("PI: " + Math.PI);
console.log("Random: " + Math.random());
console.log(Math.pow(2, 10));
console.log(Math.max(7, 9, 2));
console.log(Math.round(7.51));
```

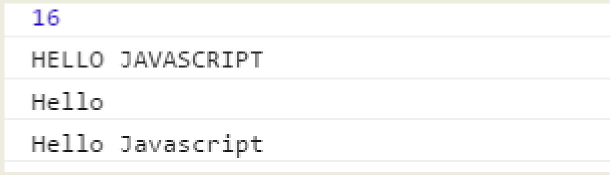


```
PI: 3.141592653589793
Random: 0.1014029317983347
1024
9
8
```

The String Class

- Used to manipulate strings.
- Note - Strings are immutable objects.

```javascript
var s = "Hello Javascript";
console.log(s.length);
console.log(s.toUpperCase());
console.log(s.substring(0,s.indexOf(" ")));
console.log(s);
```



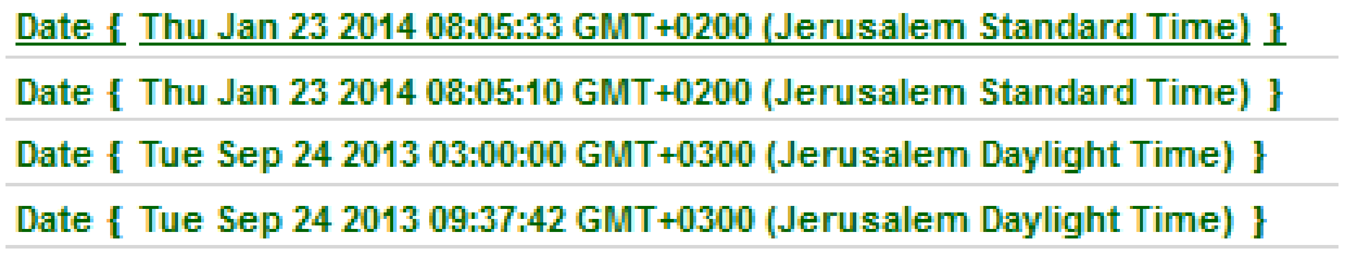
```
16
HELLO JAVASCRIPT
Hello
Hello Javascript
```

- See also: charAt, replace, split, trim

The Date Class

- Several ways for creating:

```
function cl(x) {
   console.log(x);
}

function say() {
   cl (new Date()); // current date and time
   cl (new Date(1390457110008)); //milliseconds since 1970/01/01
   cl (new Date('2013-09-24'));     // from string
   cl (new Date(2013, 8, 24, 9, 37, 42, 999)); // explicit
}
```



```
Date { Thu Jan 23 2014 08:05:33 GMT+0200 (Jerusalem Standard Time) }
Date { Thu Jan 23 2014 08:05:10 GMT+0200 (Jerusalem Standard Time) }
Date { Tue Sep 24 2013 03:00:00 GMT+0300 (Jerusalem Daylight Time) }
Date { Tue Sep 24 2013 09:37:42 GMT+0300 (Jerusalem Daylight Time) }
```

coding_
academy

Manipulating Dates

- The following code checks whether my birthday already occurred in the current year.

```
var now = new Date();
var myBirthday = new Date();
var inOneWeek = new Date();

myBirthday.setFullYear(now.getFullYear(), 8, 24);
inOneWeek.setDate(now.getDate()+5);

if (now > myBirthday) {
   alert('After Birthday');
} else if (inOneWeek > myBirthday ) {
   alert('Get Ready, birthday in 5 days');
} else {
   alert('Before Birthday');
}
```

About NaN

- Some functions return a special value called *NaN* (Not a Number) – which means that the argument passed to them cannot be evaluated to a number.
- *parseInt*() and *parseFloat*() are such functions.
- Values can be tested to see if they are *NaN* by using the *isNaN*() function.

typeof

Have a look at the different types in Javascript

```
//Numbers
typeof 12 === 'number';
typeof 3.14 === 'number';
typeof Infinity === 'number';
typeof NaN === 'number'; // Despite being "Not-A-Number"
typeof Number(1) === 'number'; // but never use this form!

// Strings
typeof '' === 'string';
typeof 'bla' === 'string';
typeof (typeof 1) === 'string'; // typeof always return a string
typeof String('abc') === 'string'; // but never use this form!

// Booleans
typeof true === 'boolean';
typeof false === 'boolean';
typeof Boolean(true) === 'boolean'; // but never use this form!

// Undefined
typeof undefined === 'undefined';
typeof blabla === 'undefined'; // an undefined variable
```

typeof

More:

```
// Objects
typeof {a:1} === 'object';
typeof [1, 2, 4] === 'object'; // use Array.isArray to differentiate regular objects
from arrays
typeof new Date() === 'object';

typeof new Boolean(true) === 'object'; // this is confusing. Don't use!
typeof new Number(1) === 'object'; // this is confusing. Don't use!
typeof new String('abc') === 'object';  // this is confusing. Don't use!

// Functions
typeof function(){} === 'function';
typeof Math.sin === 'function';

typeof null === 'object'; // This stands since the beginning of JavaScript
```

Objects (POJOs)

When we build apps we will recognize entities and use objects to describe them:

```
var customer = {
  fullName: 'Muki Ben David',
  level: 4
};

var product = {
  name: 'Sony Double Cassette',
  price: 17.80,
  features: ['Loud', 'Elegant']
};
var movie = {
  name: 'Fight Club',
  actors: [{ name: 'Brad Pitt', salary: 10000 }]
};
```

Objects

Lets play some more:

```
var movie = { name:     'Fight Club',
              actors:   [{ name: 'Brad Pitt', salary: 10000 }]
            };

movie.name += '!';

movie.stars = 5;
movie['stars'] = 6;

movie.actors.push({name: 'Edward Norton', salary: 10000});

movie  = null;
```

{coding_
academy

Object as a Map

Sometimes we will use an object for mapping a prop to a value:

```
var bestLanguageVotesMap = {
   c : 3,
   cpp: 5,
   python : 45,
   javascript: 52
};
```

- In those cases it is sometimes needed to loop through the props of the object, for this we will use the for-in loop (next slide)

Object as a Map

Note that prop names has *kind-of* the same rules as variable names, but we can use the array syntax to access any prop

```
var bestLanguageVotesMap = {
   c : 3,
   python : 45,
   javascript: 52
};

bestLanguageVotesMap['c++'] = 9;
```

One more thing: Object.keys() is a way to get all keys as an array

coding_
academy

Object converted to string

See what happen when we *force* an object to be a string:

```
var player = {
   score : 98,
   name: ’Puki’
};

console.log(player + ”);
```

For..in loop

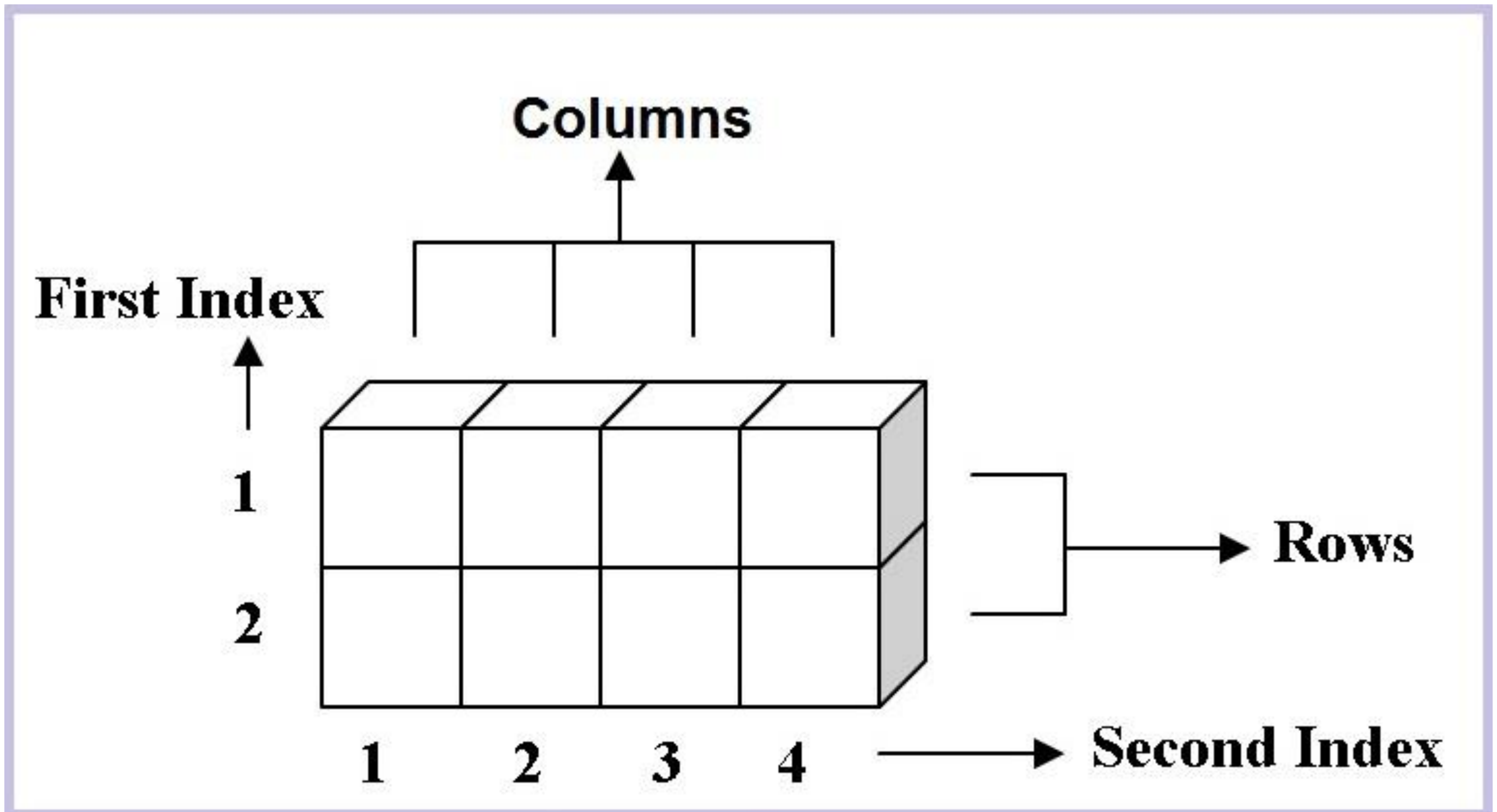
- Use this loop to go through an array or through all the properties of an object.

- It is possible, BUT
  do not the use *for...in* to loop through an array:

```
var pets = [];
pets[0] = "Chipi";
pets[1] = "Bobi";
pets[2] = "Charli";

// so this is possible but NOT a good practice:
for (var i in pets)  {
  console.log(pets[i]);
}
```

two-dimensional array

Here is an example:

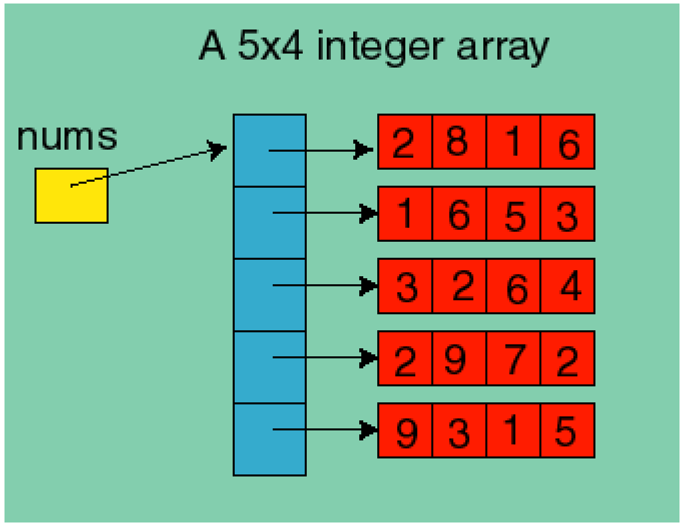


2-D Array with 8 Elements

two-dimensional array

When dealing with arrays in general, the variable is just a pointer (AKA reference) to the array.

A two-dimensional array is an array of pointers to the arrays holding the data!



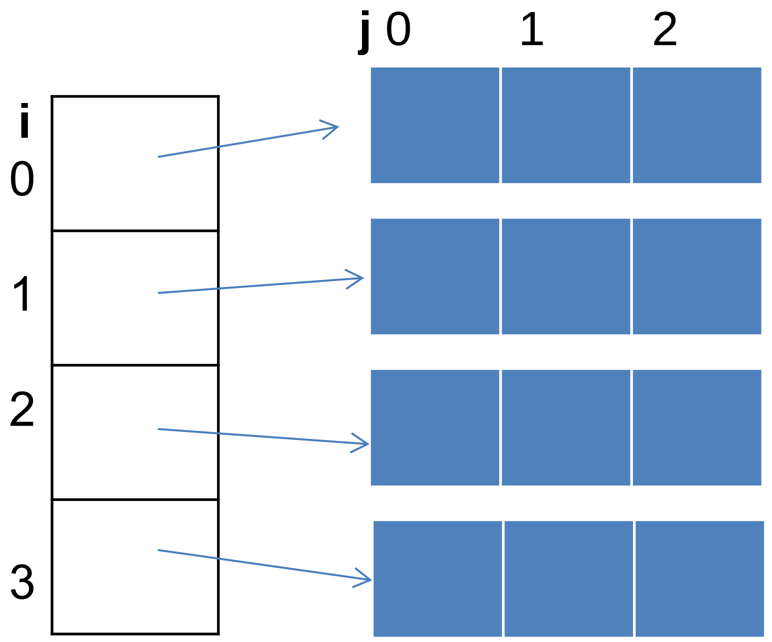
So if I need to keep an x-mix-drix board I will actually have 4 arrays, each with 3 cells.

See the next slide then come back and see if you got it.

two-dimensional array

When dealing with data structures, it is important to know how to draw them when planning your algorithm.

Multi dimensional arrays are best drawn (and imagined) like so:



```
var board = [];
for (var i =0; i< 4; i++) {
    board[i] = [];
    for (var j =0; j< 3; j++) {
        board[i][j] = '';
    }
}
```

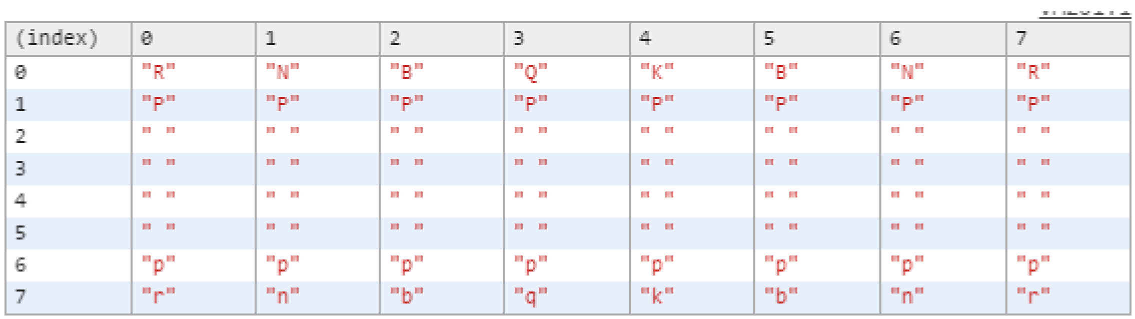
Those arrows are called pointers (or references) and they are actually simple variables holding a **memory address**

two-dimensional array

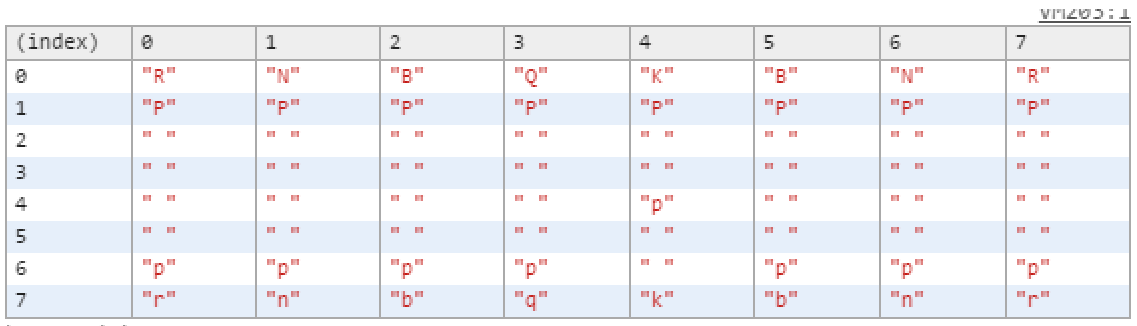
Here is a chess board as a two dimensional array of strings.

The first move is made by copying the 'p' in (6,4) to (4,4). The old position (6,4) is made blank.

```
var board = [
    ['R','N','B','Q','K','B','N','R'],
    ['P','P','P','P','P','P','P','P'],
    [' ',' ',' ',' ',' ',' ',' ',' '],
    [' ',' ',' ',' ',' ',' ',' ',' '],
    [' ',' ',' ',' ',' ',' ',' ',' '],
    [' ',' ',' ',' ',' ',' ',' ',' '],
    ['p','p','p','p','p','p','p','p'],
    ['r','n','b','q','k','b','n','r'] ];

console.table(board);

// Move King's Pawn forward 2
board[4][4] = board[6][4];
board[6][4] = ' ';
console.table(board);
```



(index)	0	1	2	3	4	5	6	7
0	"R"	"N"	"B"	"Q"	"K"	"B"	"N"	"R"
1	"P"	"P"	"P"	"P"	"P"	"P"	"P"	"P"
2	" "	" "	" "	" "	" "	" "	" "	" "
3	" "	" "	" "	" "	" "	" "	" "	" "
4	" "	" "	" "	" "	" "	" "	" "	" "
5	" "	" "	" "	" "	" "	" "	" "	" "
6	"p"	"p"	"p"	"p"	"p"	"p"	"p"	"p"
7	"r"	"n"	"b"	"q"	"k"	"b"	"n"	"r"



VM205:1

(index)	0	1	2	3	4	5	6	7
0	"R"	"N"	"B"	"Q"	"K"	"B"	"N"	"R"
1	"P"	"P"	"P"	"P"	"P"	"P"	"P"	"P"
2	" "	" "	" "	" "	" "	" "	" "	" "
3	" "	" "	" "	" "	" "	" "	" "	" "
4	" "	" "	" "	" "	"p"	" "	" "	" "
5	" "	" "	" "	" "	" "	" "	" "	" "
6	"p"	"p"	"p"	"p"	" "	"p"	"p"	"p"
7	"r"	"n"	"b"	"q"	"k"	"b"	"n"	"r"

Tip: console.table()

Handson

- Generate the multiplication table in a 2d array
- Write a function that returns the maximum value in a 2d array.
- Solve the sparse matrix challenge (see doc)

- Solve the SimCity challenge (see doc)
- Solve the Latin Square challenge (see doc)

Lets Discuss Complexity

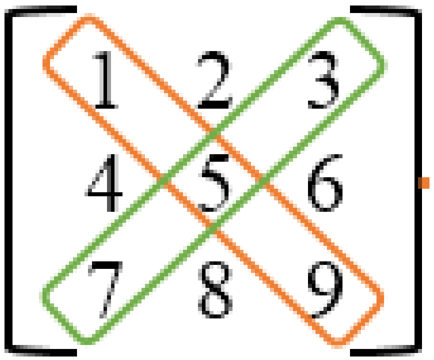
- Looping an array with size *N* has the complexity of *O(N)*
  - Examples: calculating max, sum, etc.
- So finding an item will also cost O(N)
- What if I know that the array is sorted?
- Then finding can be done using a *binary-search*
  - Its usually implemented with recursion
  - Lets see the basic idea of recursion with factorial
  - Lets review the binarySearch function

Lets Discuss Sorting

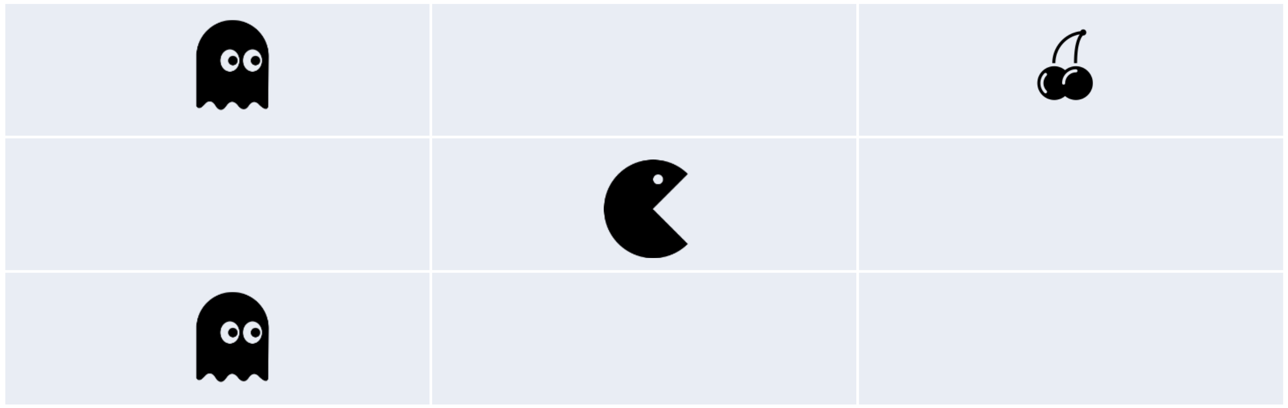
- Introducing Bubble sort
  - The Idea
  - Lets review the implementation
  - Complexity: O(N^2)
- There are many sorting algorithms
- Some of them are implemented using recursions
  - Overview the merge-sort solution

More Matrix Stuff

- Primary and Secondary diagonals



- Neighbors



Numerical Bases

0123456789

٠١٢٣٤٥٦٧٨٩

I II III IV V VI VII VIII IX X

০১২৩৪৫৬৭৮৯

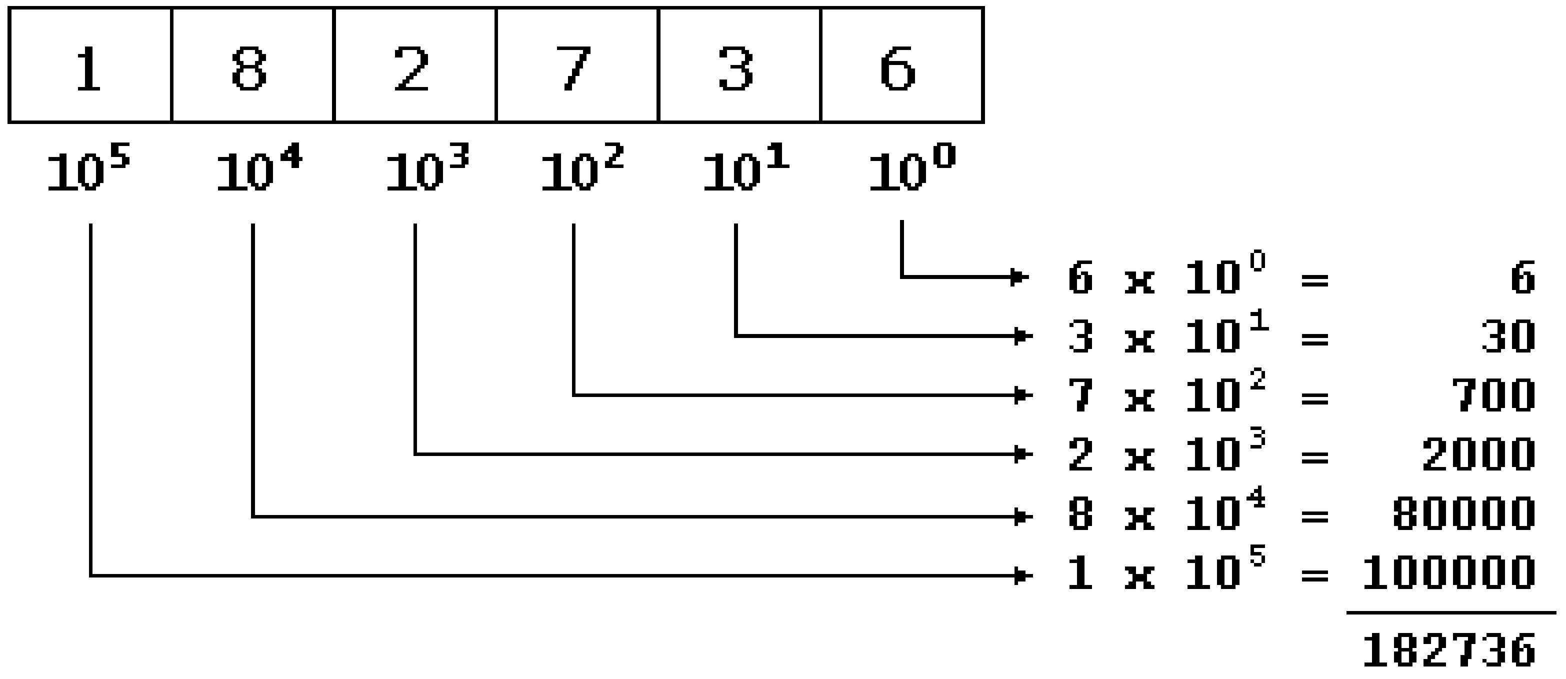
൦൧൨൩൪൫൬൭൮൯

๐๑๒๓๔๕๖๗๘๙

〇一二三四五六七八九

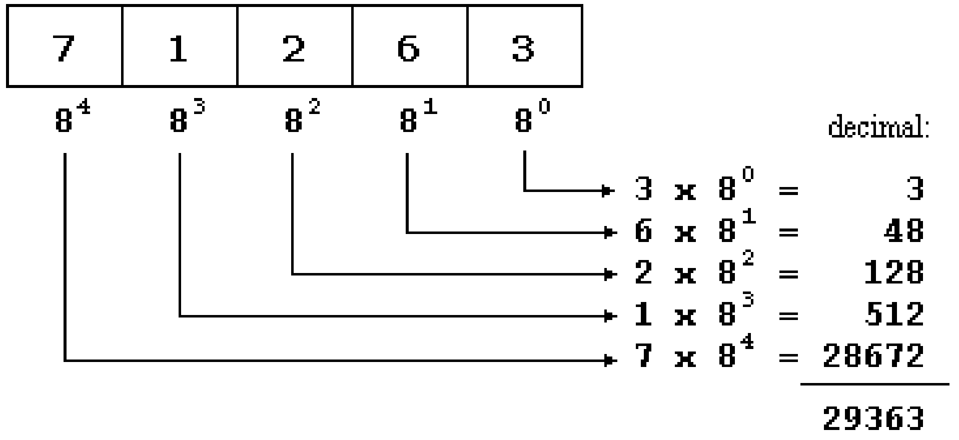
Numerical Bases

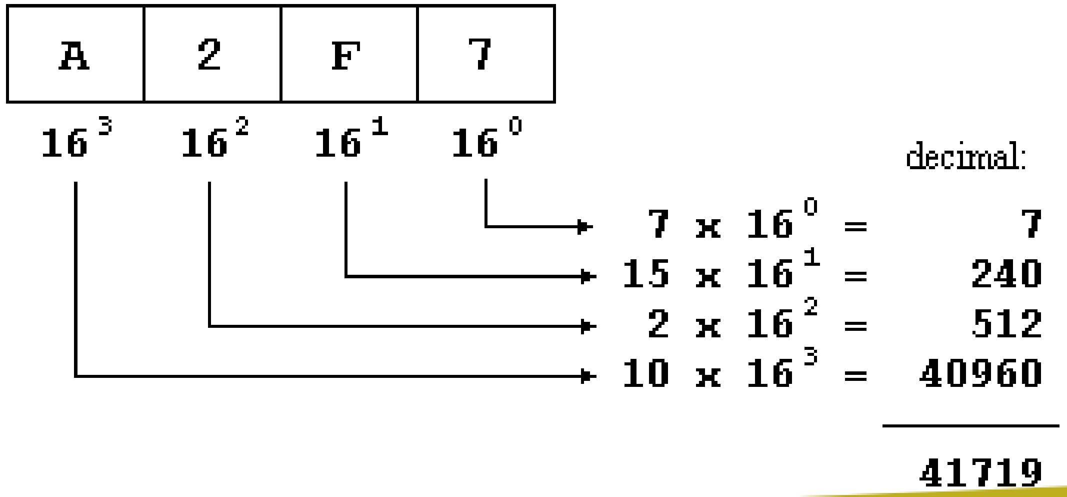
Here is how we build a number in base 10:



Numerical Bases

Here is how we build a number in other bases:

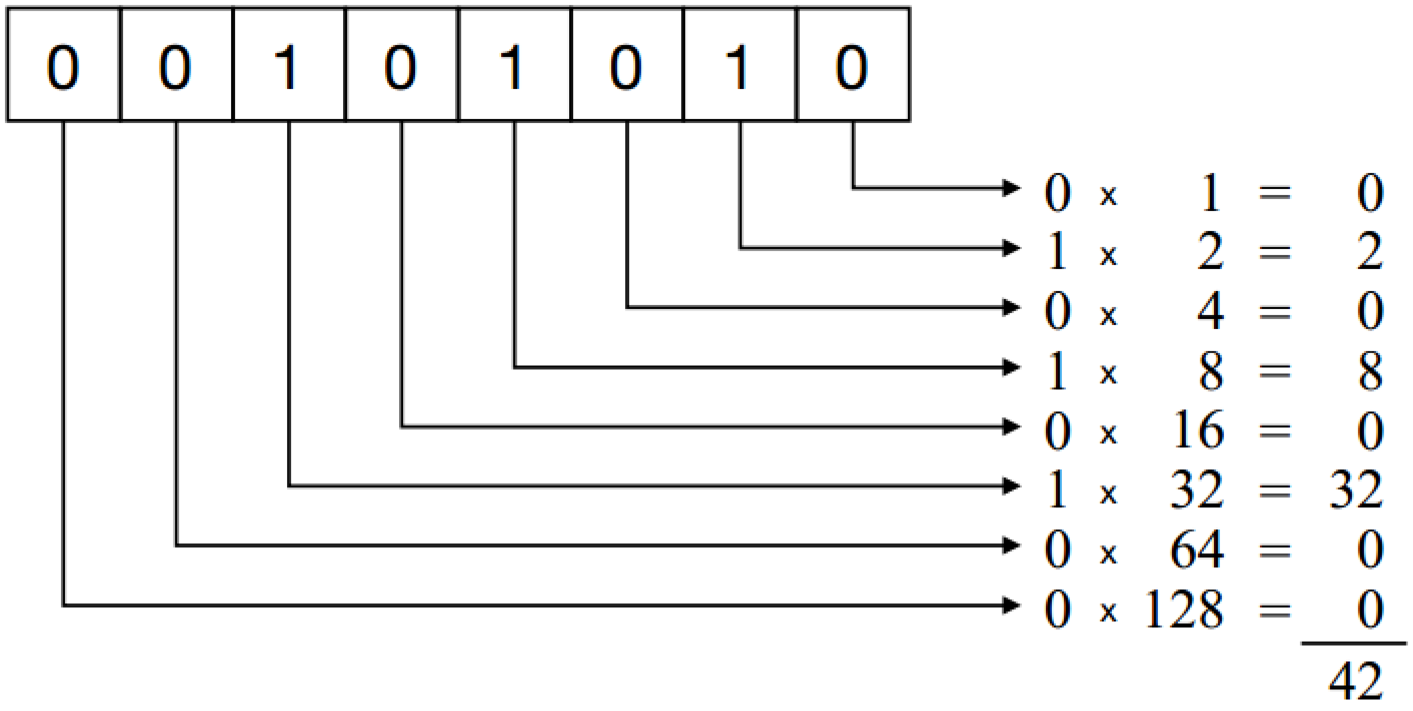




coding_ academy

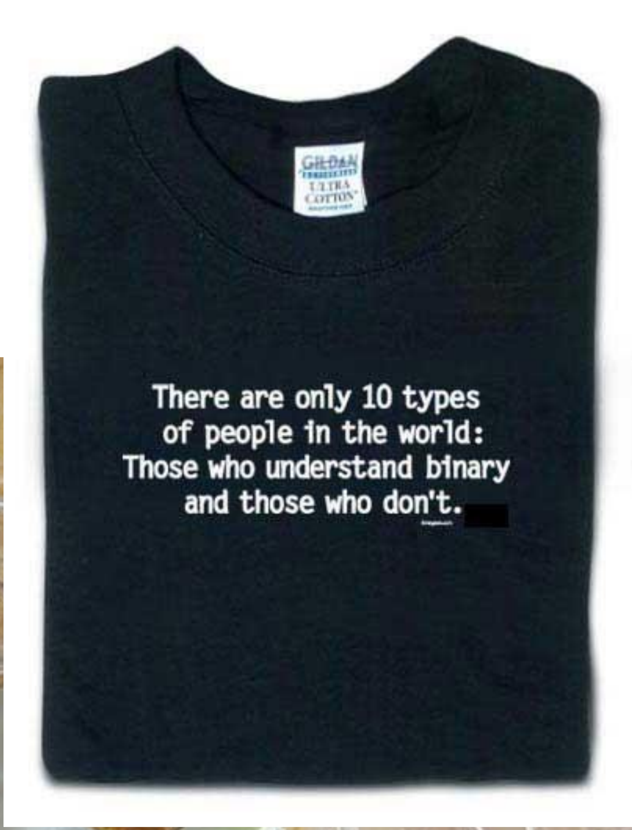
Numerical Bases

Here is how we build a number in base 2:



Base 2

Dor is
00011110
29.10.17


There are only 10 types
of people in the world:
Those who understand binary
and those who don't. [redacted]

Converting Decimal to Hexadecimal

Here are the steps:

1. Divide the decimal number by 16. Treat the division as an integer division.
2. Write down the remainder (in hexadecimal).
3. Divide the result again by 16. Treat the division as an integer division.
4. Repeat step 2 and 3 until result is 0.
5. The hex value is the digit sequence of the remainders from the last to first.

Convert the number **256** DECIMAL to HEXADECIMAL

DIVISION	**RESULT**	**REMAINDER (in HEX)**
256 / 16	16	0
16 / 16	1	0
1 / 16	0	1
ANSWER		100

{coding_ academy

More Examples

Convert the number **188** DECIMAL to HEXADECIMAL

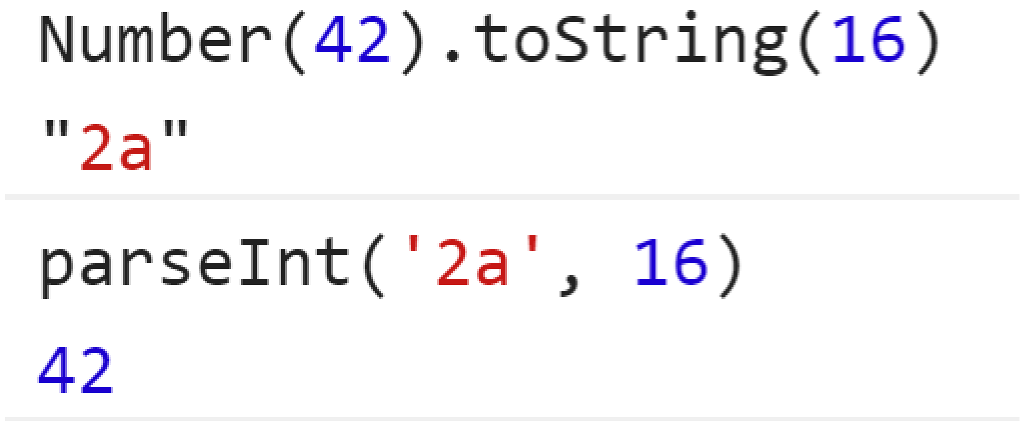
DIVISION	**RESULT**	**REMAINDER (in HEX)**
188 / 16	11	C (12 decimal)
11 / 16	0	B (11 decimal)
ANSWER		BC

Convert the number **590** DECIMAL to HEXADECIMAL

DIVISION	**RESULT**	**REMAINDER (HEX)**
590 / 16	36	E (14 decimal)
36 / 16	2	4 (4 decimal)
2 / 16	0	2 (2 decimal)
ANSWER		24E

Switching between bases

Is easy…



```
Number(42).toString(16)
"2a"
```

```
parseInt('2a', 16)
42
```

Variadic Functions

When inside a function, we can access the function’s arguments using the special parameter *arguments*. This helps when creating variadic functions:

```
// function that receives an unknown parameters count  and returns the maximum:
function myMax() {
    var max = -Infinity;
    for (var i =0; i< arguments.length; i++) {
        if (arguments[i] > max) max = arguments[i];
    }
    return max;
}
console.log('Expecting: -Infinity', myMax());
console.log('Expecting: 0', myMax(0, 0));
console.log('Expecting: 11', myMax(9, 11, 7, 1));
```

Anonymous functions

- The use of anonymous functions in javascript is very common.
- See the following examples:

```
var foo = function(name) {alert('foo ' + name);};
foo('me');

window.setTimeout(function(){alert("Time's up!")}, 3000);

setTimeout(function printTimeOut(){
    console.log('Time Out!');
}, 1000);

printTimeOut(); // Error: unknown function
```

Array functions

The *forEach* function:

```
// Instead of:
for (var i = 0, n = players.length; i < n; i++) {
    players[i].score++;
}
// You can do:
players.forEach(function(player) {
    player.score++;
});
```

Array functions - filter

The *filter* function:

```
var bigNums = nums.filter(function(num) {
   // return true if the element should be kept in the new
array
   return num > 8;
});
```

Array functions – map

The *map* function:

```
var names = [‘a’, ‘  b ’, ‘  c ’];

// Instead of:
var trimmed = [];
for (var i = 0, n = names.length; i < n; i++) {
   trimmed.push(names[i].trim());
}

// You can do:
var trimmed = [];
names.forEach(function(s) {
   trimmed.push(s.trim());
});

// But better do:
var trimmed = names.map(function(s) {
   return s.trim();
});
```

Array.some() and Array.every()

When you need to break out of a “foreach” loop, use some() or every():

```
function areAllPrimes(nums) {
    return  nums.every(isPrime);
}
// Here is a possible implementation of every:
function myEvery(arr, func) {
    var isEveryPassed = true;
    for (var i =0; isEveryPassed && i< arr.length; i++) {
        if (!func(arr[i], i, arr)) isEveryPassed = false;
    }
    return isEveryPassed ;
}
```

Array.reduce

Reduce is a very useful method of arrays, it is able to convert the array to something else. For example we can sum an array using reduce:

```
var grades = [89, 56, 100, 56];
var sumIs = grades.reduce(function(accumulator, grade){
    return accumulator + grade;
}, 0);
console.log('Sum is: ', sumIs);
```

Array.reduce

Reduce can convert an array, to an object:

```
function getElectionMap(votes) {
    var electionMap =
        votes.reduce(function(accumulator, vote) {
            if (accumulator[vote] >= 1) accumulator[vote]++;
            else accumulator[vote] = 1;

            return accumulator;

        }, {});
    return electionMap;
}
var res = getElectionMap(['Clinton', 'Trump', 'Clinton', 'Trump', 'Trump']);
```

Array.reduce

Reduce can be used to implement many other functions, such as *map* or *filter*:

```
// Implement map() with reduce()
newPrices = prices.reduce(function(acc, price){
    acc.push(price * 1.1);
    return acc;
}, []);

// Implement filter() with reduce()
var expensives = prices.reduce(function(acc, price) {
    if (price > 80) acc.push(price);
    return acc;
}, []);
```

{coding_ academy

Victorious!

You have successfully grasped the basics of

Javascript

as a programming language

{coding_
academy