Hello

Javascript

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

Javascript is Everywhere!

- Javascript is the only language spoken by the browsers.
- Javascript is used to create:
  - Web, Mobile and Desktop applications
  - It is used both in client side and server side

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>)
- Javascript (like most programming languages) is a case sensitive language.

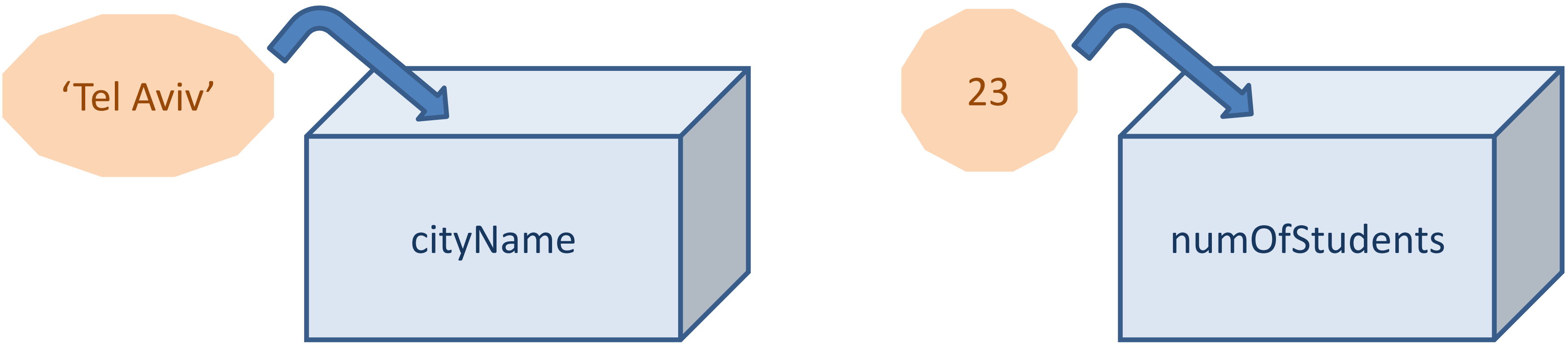
The Seven tools of the Developer

1. Variables
2. Expressions
3. Conditions
4. Loops
5. Functions
6. Arrays
7. Objects

Variables

A variable is a piece of memory that stores some value

Variable is like a box with a name tag,
holding inside a piece of data:



Variables

We define variables before using them using the *var* keyword (more options later)

```
var age = 18;
var name = 'Puki';
var isHappy = true;

alert(name);
```

Data types

The basic data types in JavaScript are:

- number (17, 0, -3.14)
- string (‘Hello There’, ‘’, ‘123’)
- 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
- We concatenate (add) strings together using the operator +

```
var name = 'Puki'
var greet = 'Welcome ' + name
```

Comments

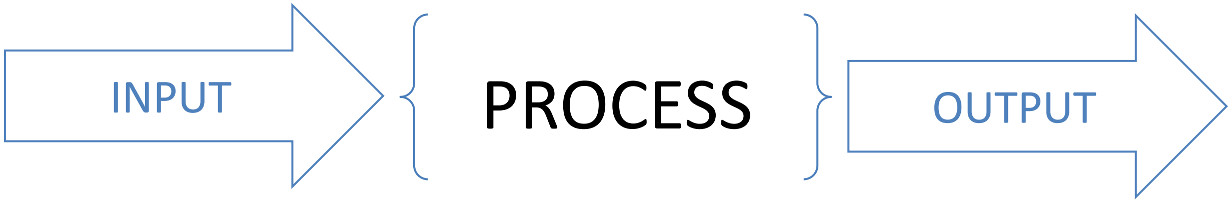
Comments are important for any programming language:

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

```
<script>
  // I am a comment, and proud of it!

  var num = 4;
  /*
     This is a multi-line
     comment.
  */

  // alert('Hello');
</script>
```

The input-process-output model

- Computer programs works by this model:
  - Receives inputs from a user or other source
  - Does some computations on the inputs
  - And returns the results of the computations



Output to the user

The *alert()* function is a simple way to output something to the user:

```
// Greet the user:
alert('Hi!')
```

Output to the console

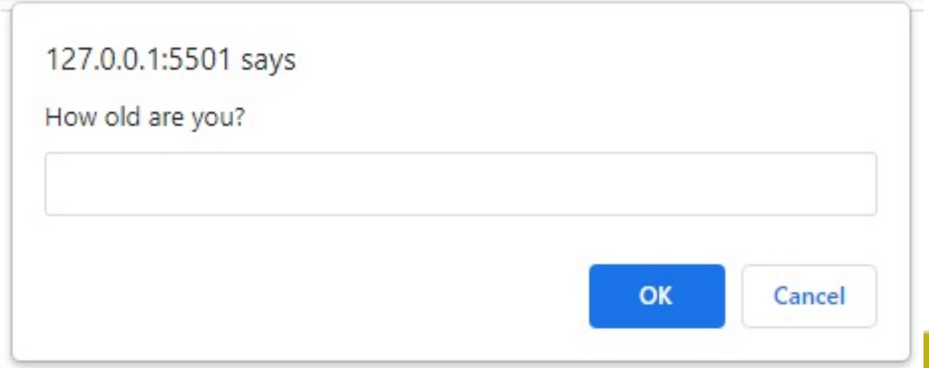
The *console.log()* function is a simple way to output something to the developer-tools:

```
// See the console in the developer-tools
console.log('Here!')
```

Reading input from the user

Lets use the *prompt()* function for getting input from the user:

```
// Get some string from the user:
var name = prompt('What is your good name?’)

// When getting numbers we convert them with +
var age = +prompt('How old are you?')
```



127.0.0.1:5501 says

How old are you?

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: +, *, %

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.

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 )
```

Booleans and Conditions

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

- Here are some boolean expressions:

  num > 7   *amount <= 100*   *count === 10*

- We use them for **conditions**

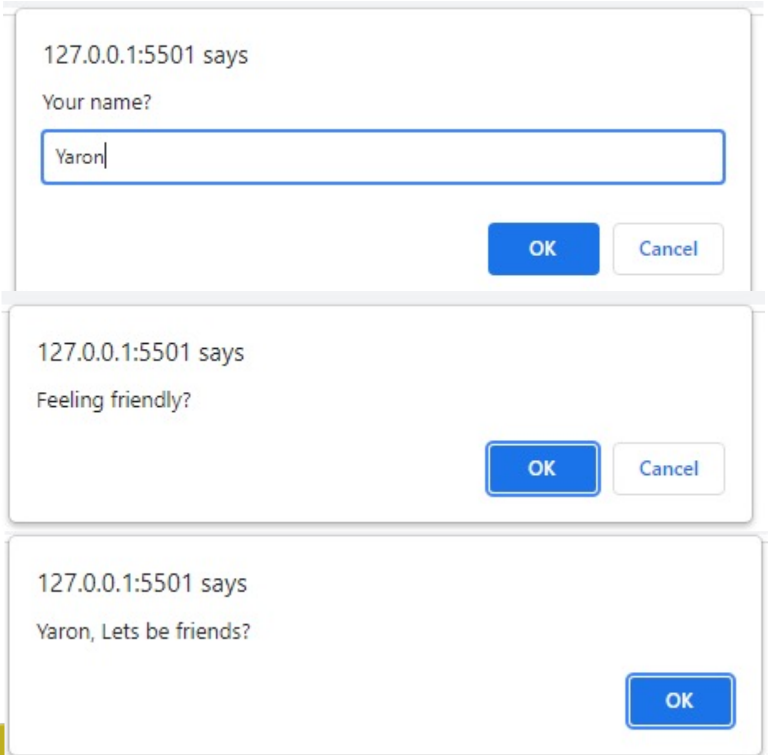
```
if (grade >= 70) {
    alert('Passed')
} else {
    alert('Failed')
}
```

The confirm() popup

There are just 3 built-in popups in the browser:

- ***alert()*** – shows a msg.
- ***prompt()*** – reads input.
- ***confirm()*** – shows a msg with Ok/Cancel buttons, and returns a boolean value (true / false)

```
var userName = prompt('Your name?')
var isFriendly = confirm('Feeling friendly?')
if (isFriendly) {
    alert(userName + ', Lets be friends?')
}
```



127.0.0.1:5501 says

Your name?

Yaron

127.0.0.1:5501 says

Feeling friendly?

127.0.0.1:5501 says

Yaron, Lets be friends?

Boolean operations

Name	Example	Result
Equal	a == b	TRUE if a is equal to b. DON’T USE because 17 == ‘17’
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 it to keep consistent
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.

Boolean operations

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

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.

Example:

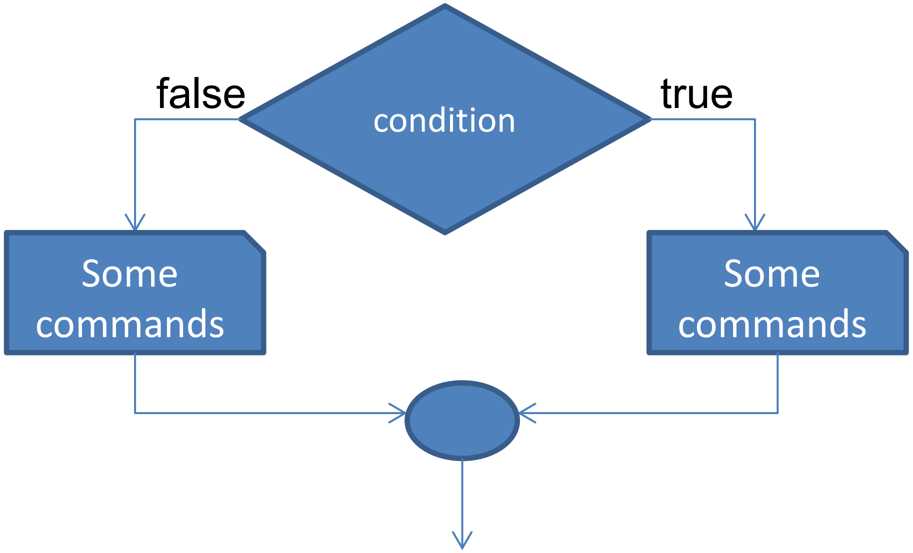
```
if (tasteScore >= 80 || (failsCount < 3 && isFeelingMercy)) {
    alert('Tip')
} else {
    alert('Thanks')
}
```

Live Coding - Conditions

- Read 2 numbers and print the bigger one
- Ask the user for the meal price, and print the total price with a 10% tip
  - Use confirm() to ask the user if he was super happy, if so, add 5% more

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 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 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

```
if (tasteScore + bonusForGoodService >= 80 || (failsCount < 3 && isFeelingMercy)) {
    alert('Tip')
} else {
    alert('Thanks')
}
```

What’s *truethy*

- false is
    - The value *false*
    - The value *null*
    - The value *0*
    - The value *undefined*
    - The empty string *“”*
    - And the number *NaN*.
- The rest are true

```
var x = '';
if (x) console.log('Truthy')
else console.log('Falsy')
```

Functions

An application is made with many functions

Functions

- A function is a portion of code which performs a specific task.
- Here is a simple function:

```
function sayHello() {
    alert ( 'Hello ' );
}

sayHello();
```

Functions helps keep our code DRY

- Functions are a way to pack some functionality in a reusable way and keep our code DRY

- DRY – Don’t Repeat Yourself

We don’t want to copy & paste in a way that creates **more and more** code

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);
```

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

- Write the function fancyLog(msg), it prints the message surrounded by stars
- Write the function:
  personalizeMsg(greet, name, balance)
  that returns something like:
  'Hi Puki, your balance is 20!'

const

- consts are used to define unchangeable vars
- Here are some examples:

```
const SYMBOL = '*'
const MSG = 'Lets go!'
const HOUR = 1000 * 60 * 60
const LETTERS = 'abcdefghijklmnopqrstuvwxyz'
const PIN_CODE = '0796'
```

Loops

- Loop is a useful flow control
- Loops enable us to do something repeatedly
- This is a *while* loop:

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

While Loop

Simple example:

```
var count = 0
while(count < 10) {
    console.log (count)
    count++
}
```

While Loop

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

Dry Run

Follow the code line by line and monitor the variables in a table:

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

num	digit	Output

Sentinel Loop (Zakif)

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 names until “QUIT” is entered then print the names separated by *

Loops – hands-on

- Read a number and check if that number is prime
  Add unit testing!

```
// Unit Testing
var num = 124;
var res = isPrime(num);
console.log('INPUT:', num , 'EXPECTED: false, ACTUAL:' + res);

num = 5915587277;
res = isPrime(num);
console.log('INPUT:', num , 'EXPECTED: true, ACTUAL:' + res);
```

- Read a 5 digits number and check if it symmetric
  Add unit testing!

Unit Testing

Unit tests are automated tests written and run by developers to ensure that a section of an application (known as the "unit") meets its design and behaves as expected.

- The basic unit we test is a function
- The output of the Unit-Testing should be readable by humans (mostly other developers)

```
Assert.AreEqual(
    GetTimeOfDay(),
    "Morning" );
```

Unit Testing

- Here is an example:

```
var num = 10;
var res = factorial(num);
console.log('factorial - INPUT: ', num,
  'EXPECTED: ', 720, 'ACTUAL: ', res);

num = 0;
res = factorial(num);
console.log('factorial - INPUT: ', num,
  'EXPECTED: ', 1, 'ACTUAL: ', res);
```

Unit Testing

- When writing tests we should also consider edge cases (Factorial is defined only for n >= 0)

```
num = -8;
res = factorial(num);
console.log('factorial - INPUT: ', num,
  'EXPECTED: ’, NaN, 'ACTUAL: ', res);
```

Functions - Check point

- Functions can have 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

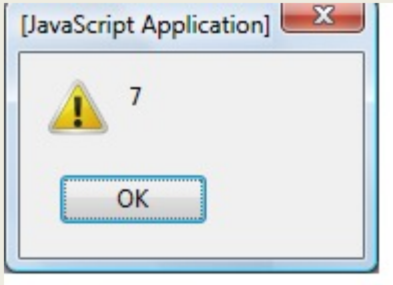
- Local variables
  - Declared inside scopes (such as a function)
  - Live from declaration to the end of the scope
- Global variables
  - Declared outside any function
  - Can be used throughout the page and inside functions
  - Live from declaration
  - Can be accessed from console

Variables’ scopes

Important JS fact:

Have a look at the following code:

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

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



7

The problem is that the max variable became global when set to 7

- Always define your variables (with var)
- *‘use strict’;* to watch your back

Strict mode

- Use *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’;
```

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 );
```

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 (hasItems() && 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.

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
}
```

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.

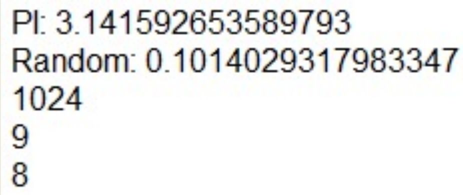
Built-in Classes

- In Javascript, we have some built-in classes :
  - String, Date, Math, 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));
```



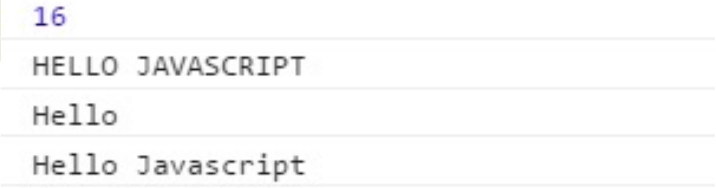
```
PI: 3.141592653589793
Random: 0.1014029317983347
1024
9
8
```

The String Class

```
var str = 'hello javascript';
str.charAt(0).toUpperCase() +
str.substring(1)
// Prints: "Hello javascript"
```

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

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



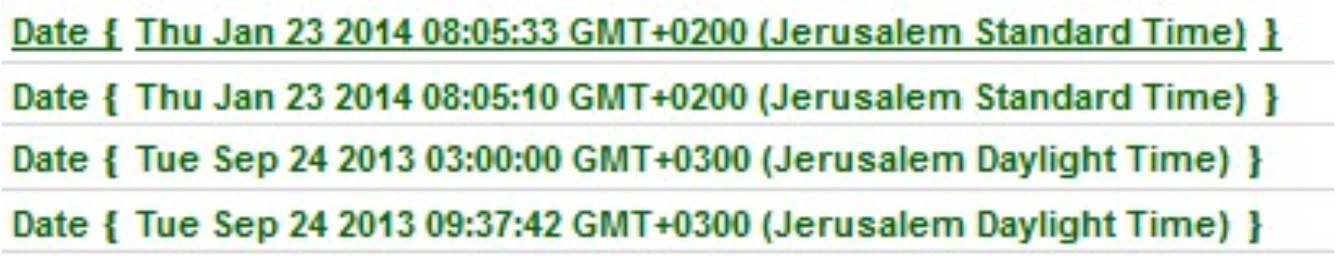
```
16
HELLO JAVASCRIPT
Hello
Hello Javascript
```

- See also: trim, replace, charAt, indexOf(‘a’, 8)

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) }
```

Manipulating Dates

- The following code checks whether my birthday already occurred in the current year.
- Note that the month starts from 0, so 8 is September.

```
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');
}
```

Timestamps!

- Timestamp is the number of milliseconds that have passed since January 1, 1970 at midnight.
- It’s a way to specify a point in time with a single number.

Handson

- Write a function formatTime(time) that returns a formatted time:
  - Just now
  - few minutes ago,
  - Today
  - Yesterday
  - At 2018-09-24 Time: 10:23

  - Add Unit Testing

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);
```

Loop over a string

```
var str = 'ABCDE';
for (var i = 0; i < str.length; i++) {
    var letter = str.charAt(i)
    console.log(letter);
}
```

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 {}

Hands-on

- Write a function: printPrimes that gets 2 numbers: minRange and maxRange and prints 10 prime numbers in that range.

Continue to next iteration

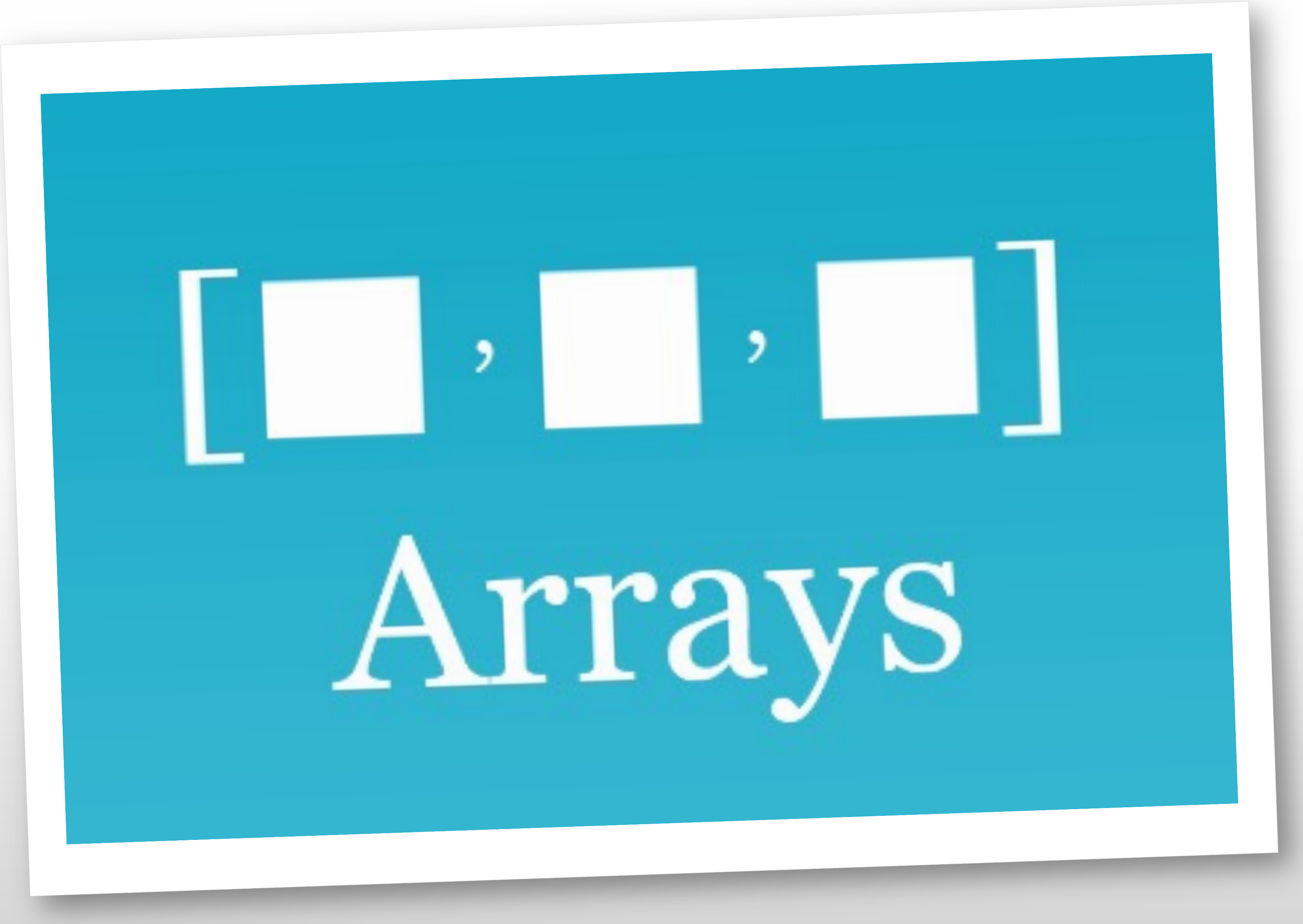
To skip the current iteration in the loop, use continue;

```
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
```

More examples

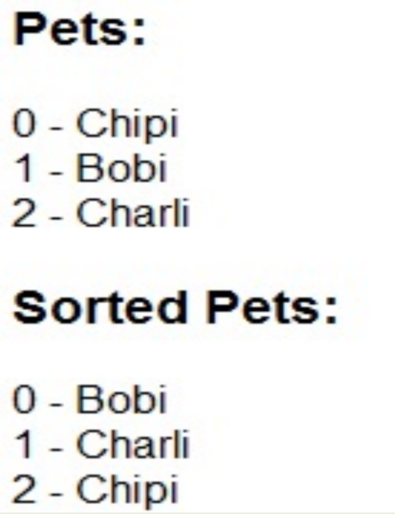
```
// 10 times, but print only odd numbers:
for (var i=1;i<=10;i++){
    if (i % 2 == 0) continue;
    console.log(i + ", ");
}

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


[□, □, □]
Arrays

Arrays

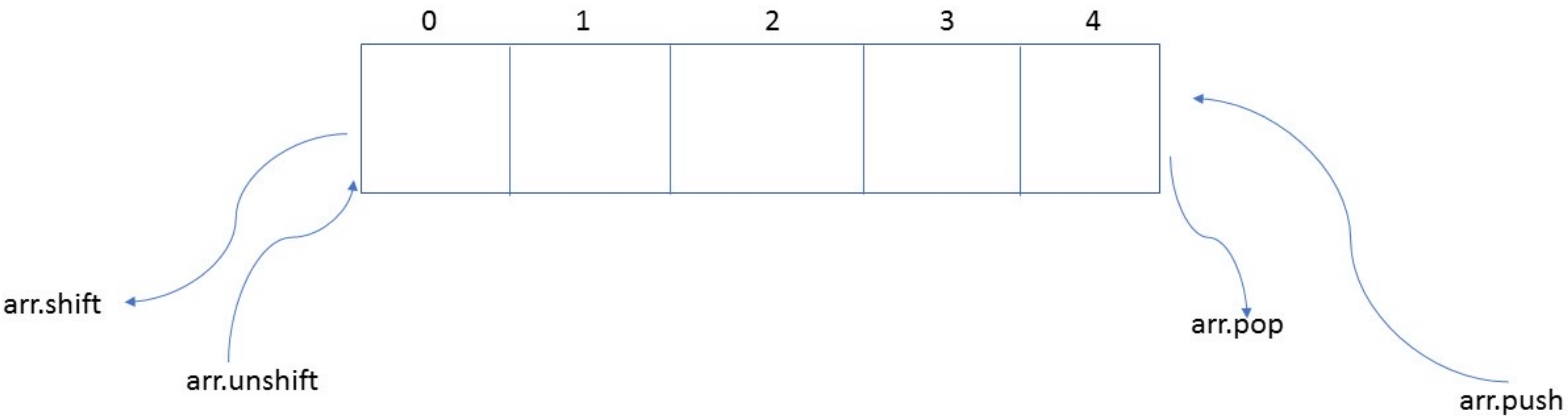
It is often needed to store multiple values in a single variable

```
// 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
```

Adding / Removing / Updating items



arr.splice(idx, 1) – remove at index

arr.splice(idx, 1, newVal) – replace at index

arr.splice(idx, 0, newVal) – Add at index

Looping through an array

Calculating score average:

```
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
```

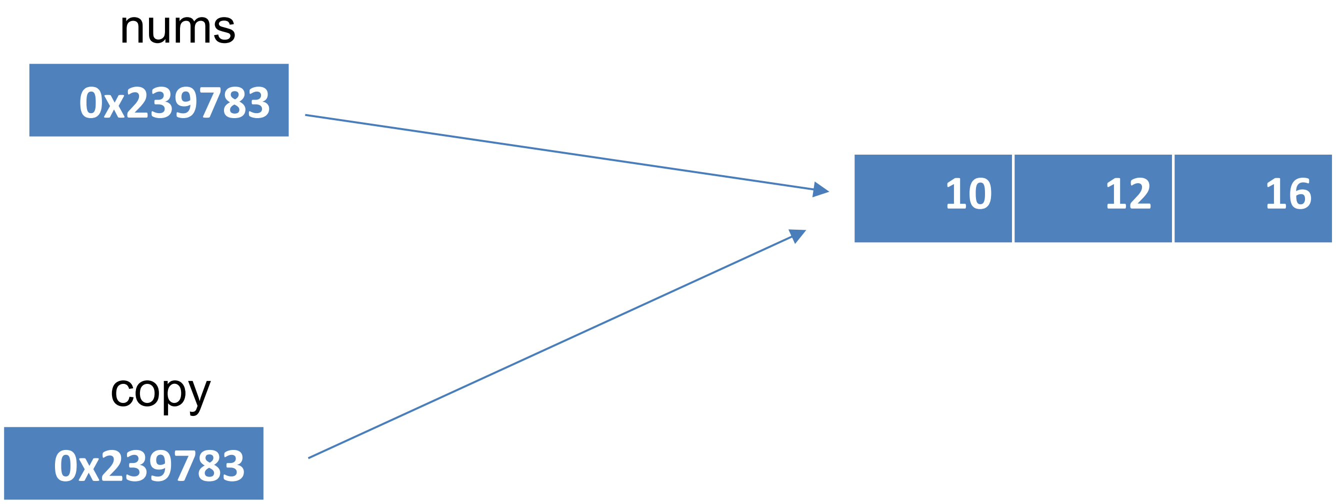
References vs Values

The array name is a reference (pointer / memory address) 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

The array name is a reference (pointer / memory address) to the beginning of the array:



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 = []
}
var vals = [101, 102];
foo(vals);
console.log(vals)
```

Arrays – hands-on

- Write a program that generate 10 random numbers, push into array and print it
- Use split & join:

```
    var names = 'Muki,Puki,Shuki'.split(',');
    var str = names.join('|')
    // str is: Muki|Puki|Shuki
```

- Use slice to copy an array:

```
var nums = [0, 7, 9, 6]
var numsCopy = nums.slice();
```

-

Arrays – hands-on

- Write the function:

  ```
  isItemExist(items, item)
  ```

- We need to store data about students, lets Use 2 arrays:

  ```
  var studentNames = ['Popo', 'Momo', 'Lolo’];
  var studentGrades = [90, 54, 100];
  ```

  - Write the function printStudentsGrades()
  - Write the function addStudent() that prompts for details
  - Write the function printBestStudentName()

Objects

```
var client = {
    fullName: 'Muki Ben David',
    balance: 426
};
```

Objects

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 }]
};
```

Play with Objects

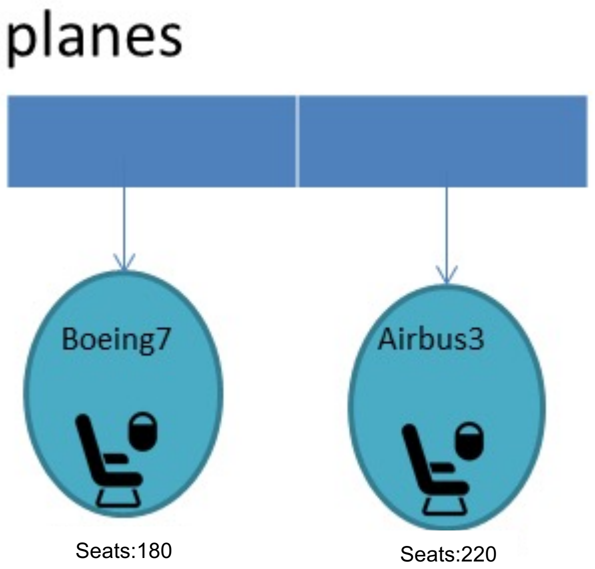
```
var movie = { name:     'Fight Club',
              actors:   [{ name: 'Brad Pitt', salary: 10000 }]
            };

movie.name += '!';

movie.stars = 5;
movie['stars'] = 6;
var key = ‘stars’;
movie[key] = 7;

movie.actors.push({name: 'Edward Norton', salary: 7000});

movie  = null;
// At this point we can no longer
// access the movie
// so it will be cleaned from memory
// by the garbage collector
```

Arrays and Objects

- When working with data structures, its useful to draw them

Note: Naming convention - using plurals for arrays

- Items, values, nums, names, students, etc
  use the word *arr* sparsely

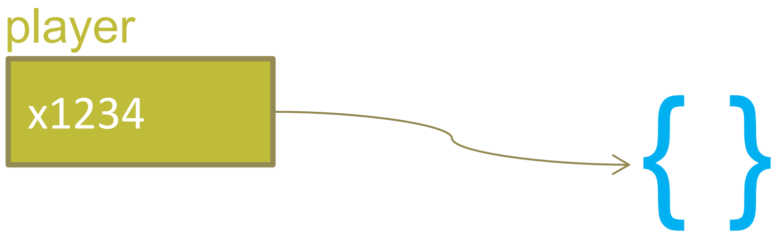


Object is a reference (pointer)

Lets understand that (just like with arrays)

player is actually a variable holding an address of an object

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



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 + ”);

// Prints: "[object Object]"
```

Objects – usually have an id

Here is a simple mechanism to generate an id on the client side:

```
var gNextId = 101;
var gPlayers = [
    { id : gNextId++, name: 'Muki'},
    { id : gNextId++, name: 'Puki'}
];
```

Note: Those Ids are usually generated at the backend by the database.

Objects – hands-on

- createPlayer(name) – creates and returns a new player object with random score (0..100)
- getPlayerById(playerId) – returns a player by its id
- findBestPlayer(players) returns the best player
- findBestPlayers(players) returns the best players

```
var player = {
  id : 101,
  score : 98,
  name: ’Puki’
};
```

Objects – hands-on 2

- Movies And Actors (see doc)
  - createMovie, createMovies
  - find actor with highest salary
  - get movies of actor. (func that return a movies array)

```
var movie = {
  id : 101,
  name : ‘Harry Potter’,
  actors:[{name: ‘Daniel Redcliffe’, salary: 9000}]
};
```

Object as a Map

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

```
var langVotesMap = {
   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)

For..in loop

Use the for-in loop to go through all the properties of an object.

Mainly used when the object is a map:

```
var langVotesMap = {c : 3, 'c#': 5, javascript: 52};
console.log(langVotesMap);

var langName = prompt('Which is your favourite language?');
var count = langVotesMap[langName];

if (langName) {
   langVotesMap[langName] = (count)? count+1 : 1;
}

for (var langName in langVotesMap ) {
     var votesCount = langVotesMap[langName];
    console.log('Language: ' + langName + ' has: ' + votesCount + ' votes');
}
```

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 langVotesMap = {
   c : 3,
   python : 45,
   javascript: 52
};

langVotesMap['c++'] = 9;
Object.keys(langVotesMap)
```

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

Objects – hands-on 3

Metushelah

- Build a data structure that represent a family tree, we might make it graphical later, but for now we focus on the data structure:
- 1)A person object that has:
  - i.an id (1001,1002, etc),
  - ii.a name,
  - iii.a birthdate (a date object)
  - iv.parents –array by size of 2
  - v.childs –array by any size

- 2)Build a data structure byId (an object) that will store all people by their ID
- 3)write functions:
  - i.addChild(toPersonId, childPerson)
  - ii.addParent(toPersonId, parentPerson)
- 4)Use those functions to create a tree with some data (i.e. your family, from Adam to Noah, etc.)
- 5)Write a function that prints the name of the parents that gave their son, the longest name.
- 6)Write a function getCousinsCount(personId) that returns number of cousins for a certai

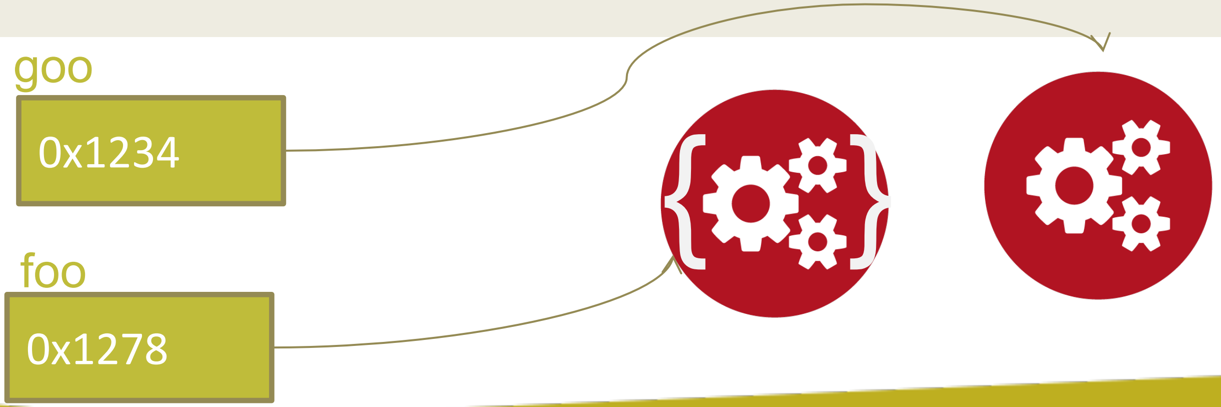
coding_
academy

Function is a reference (pointer)

Just like with arrays or objects,

foo is actually a variable holding an address of a function

```
function foo() { console.log('foo'); }
var goo = foo;
goo();

foo = function (name) { console.log('foo ' + name); };
foo('me');
```



goo

0x1234

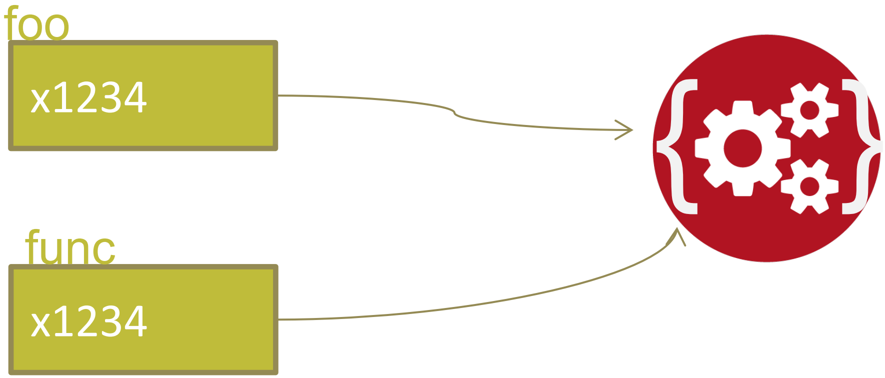
foo

0x1278

So function is just a reference

Here we pass the address of *foo* to another function that runs it:

```
function runThisFunc(func) {
    console.log('Proud to run: func()');
    func();
}

function foo() { console.log('foo'); }
runThisFunc(foo)
```



Anonymous functions

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

```
setTimeout(function () { alert('Times up!') }, 3000);


var players = [{ score: 82, name: 'Muki’ },
               { score: 96, name: 'Puki' }];

players.sort(function (p1, p2) { return p1.score - p2.score })
```

Arrow functions

- Arrow function is a newer syntax to define anonymous functions
  - We will later discuss them in deep
- Lets see how it looks:

```
setTimeout(function () { alert('Times up!') }, 3000);
setTimeout(() => { alert('Times up!') }, 3000);


var players = [{ score: 82, name: 'Muki’ },
               { score: 96, name: 'Puki' }];

players.sort(function (p1, p2) { return p1.score - p2.score })
players.sort( (p1, p2) => { return p1.score - p2.score })
```

Default function parameters

- Default parameters allow parameters to be initialized with default values if no value (or undefined) is passed.
- See:

```
function mult(num1 = 6, num2 = 1) {
    return num1 * num2;
}

console.log(mult(5, 2));
// result: 10

console.log(mult(5));
//  result: 5

console.log(mult());
//  result: 6

console.log(mult(undefined, 10));
//  result: 60
```

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
```

two-dimensional array

An array of which every item is an array (of values):

```
var data = [[12, 21, 32], [11, 89], [1, 2, 3, 4]]
```

We will focus on *special* 2d arrays that are called matrixes: all rows have the same length.

```
var data = [[73, 27],
            [11, 89],
            [56, 44]]
```

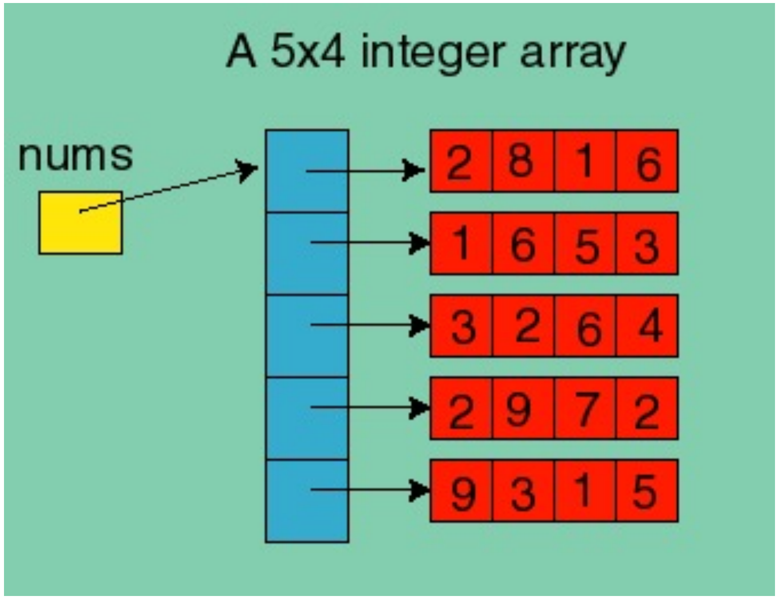
Rows, Cols, Cells

73	27
11	89
56	44

Matrix in memory

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

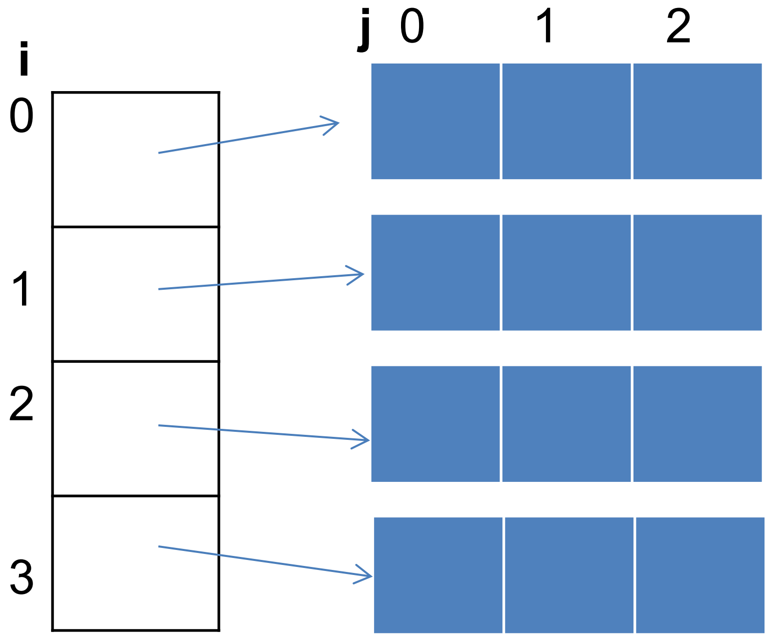
A two-dimensional array is an array of pointers to the arrays that hold the data



Note how the blue array is an extra array holding the matrix rows

The matrix

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 pointers (AKA references) and they are simple variables each holding a **memory address of an array –** a row in the mat.

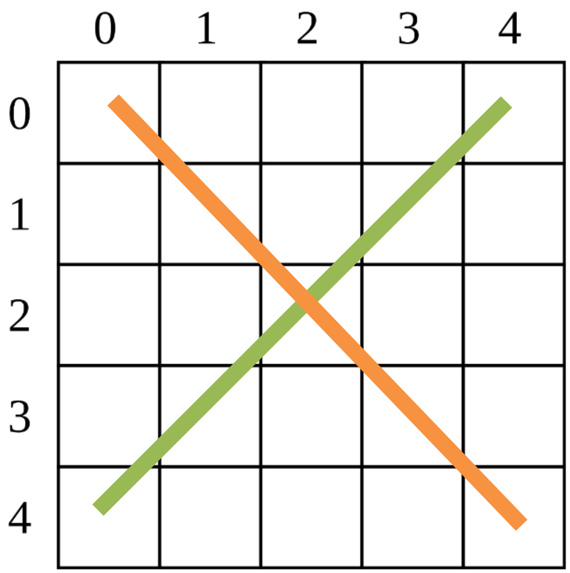
Handson

- Generate the **multiplication table** in a 2d array
- Write a function that returns the **maximum** value in a 2d array.

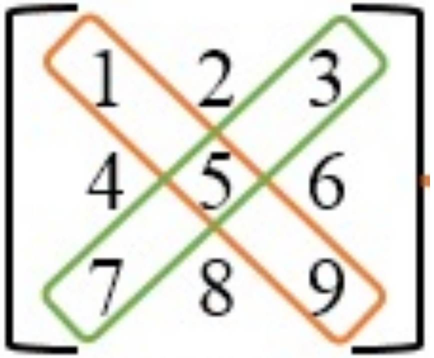
- Solve the following challenges:
  - Sparse matrix
  - SimCity
  - Latin Square

Square Matrix

When the matrix is square (`mat.length === mat[0].length`) we have diagonals:

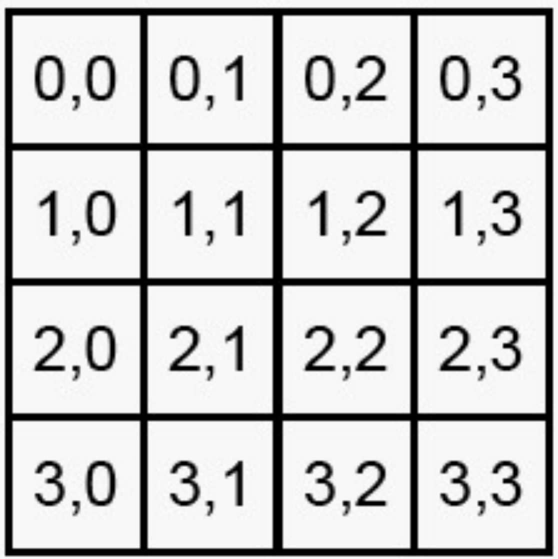


Primary and Secondary diagonals



```
function printPrimaryDiagonal(squareMat) {
    for (var d = 0; d < squareMat.length; d++) {
        var item = squareMat[d][d];
        console.log(item);
    }
}

function printSecondaryDiagonal(squareMat) {
    for (var d = 0; d < squareMat.length; d++) {
        var item = squareMat[d][squareMat.length - d - 1];
        console.log(item);
    }
}
```

Note: as **d** grows the column index gets smaller



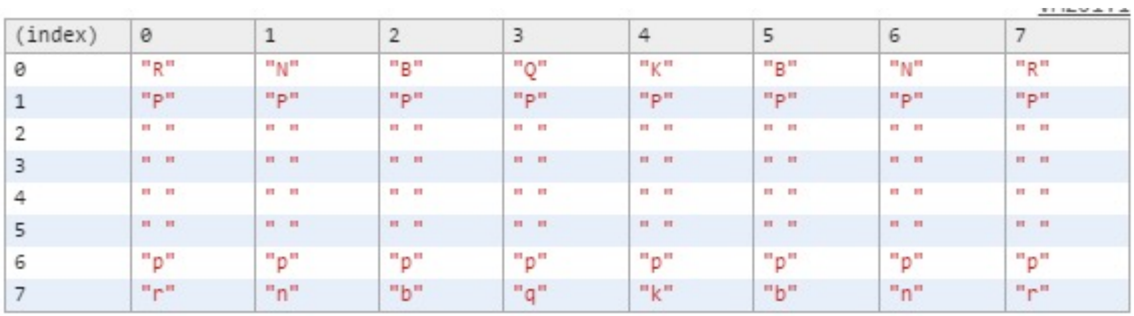
0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Matrix as a game board

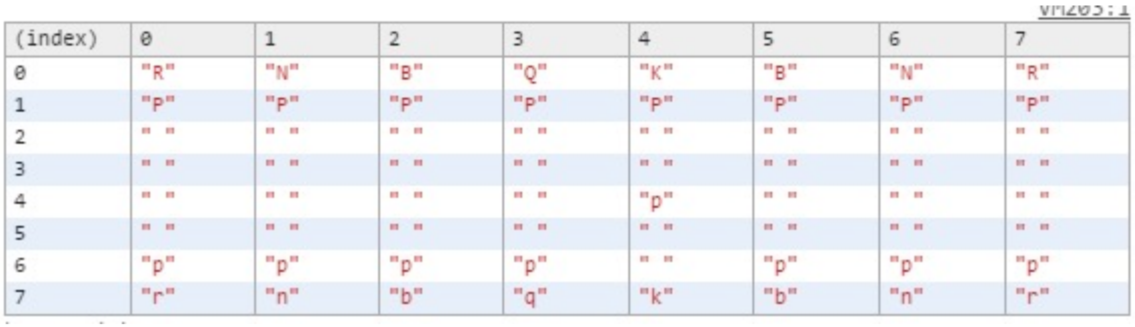
Here is a chess board represented as a matrix of strings.

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



VM263:1

(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()

Let's Plan X-Mix-Drix

Let’s identify global data names and structures

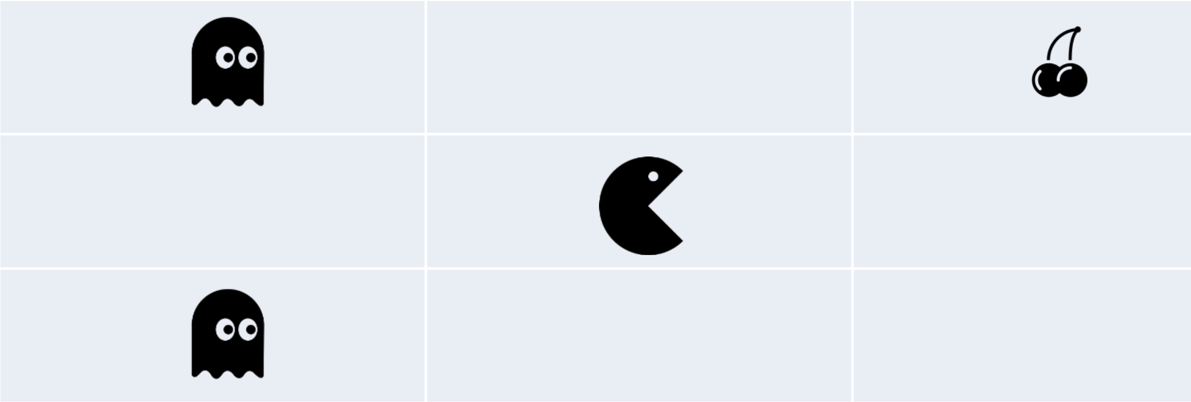
X-Mix-Drix

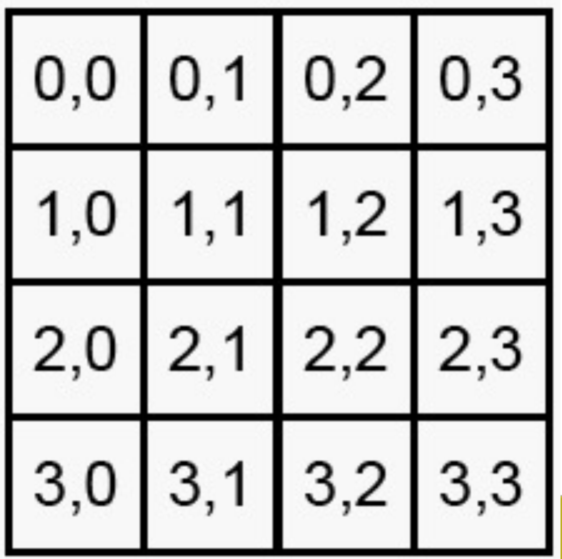
```
gBoard = createBoard()

init()
  playGame()

playUserMove()
   getPos(strPos)

playComputerMove()
   findEmptyPos()

playMove()
   isVictory()
```

Matrix Neighbors

When dealing with matrixes, it is sometimes needed to look around a certain cell

- Generally, cells have 8 neighbors
- Cells at the edges have less..

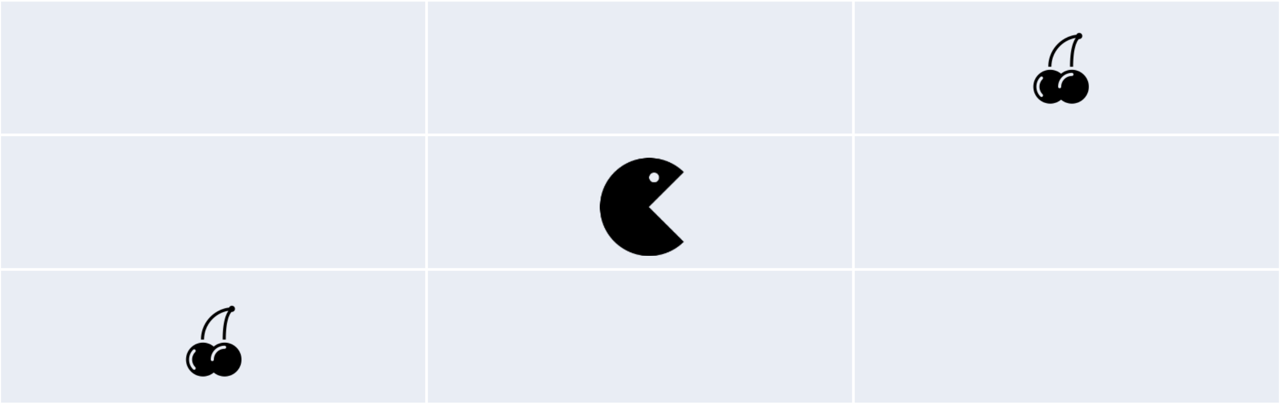




0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Matrix Neighbors

Code the function countFoodAround(mat, rowIdx, colIdx)



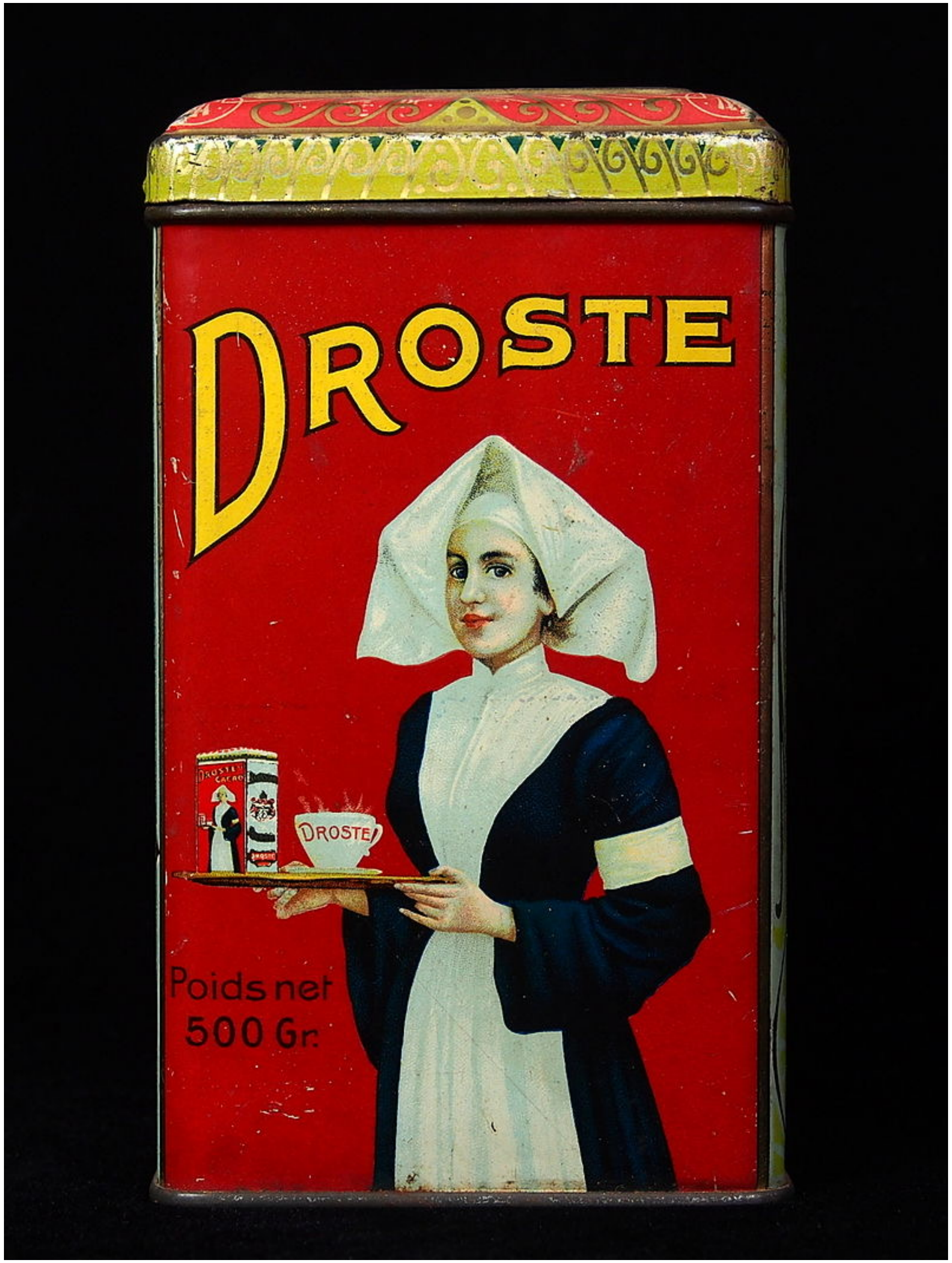
typeof

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
```

Introduction to Recursion

Recursion occurs when a thing is defined in terms of itself or of its type.



When a function calls itself

Review the following recursion

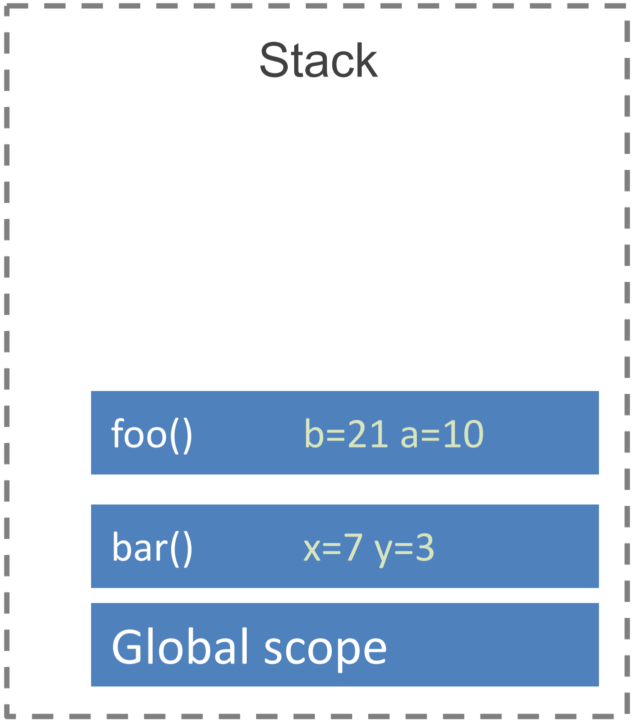
Here, a global variable is used to create a loop without using an actual loop

```
var gCount = 10;
foo();
function foo() {
  gCount--;
  if (gCount === 0) return;
  foo();
}
```

The call-stack

Lets review the call-stack and examine how functions get executed

```
function foo(b) {
    var a = 10;
    return a + b + 11;
}

function bar(x) {
    var y = 3;
    return foo(x * y);
}

console.log(bar(7));
```



Stack

foo() b=21 a=10

bar() x=7 y=3

Global scope

When a function calls itself

Review the following recursion

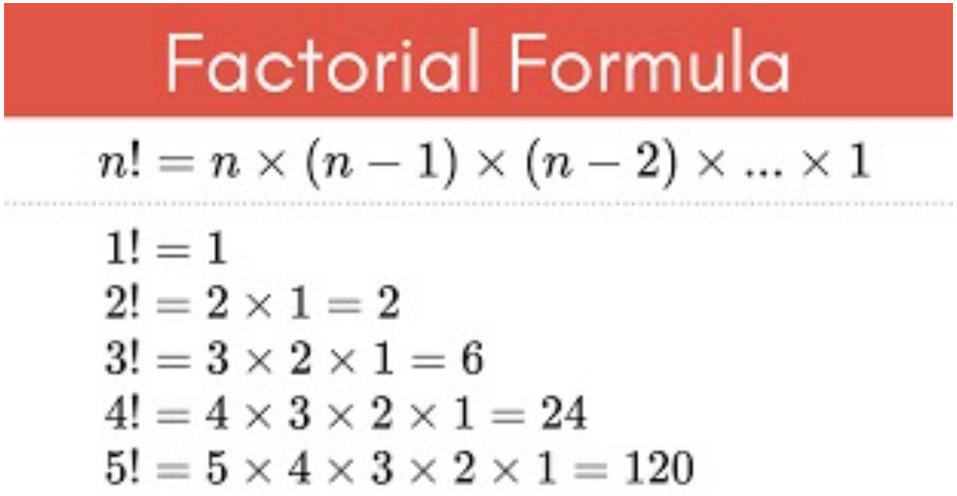
```
foo(10);
function foo(count) {
    if (count === 0) return;
    console.log('Foo!', count);
    foo(count-1);
}
```

Lets review some code samples

Factorial:

n! = n * (n-1)!   1! = 1

Review the function factorial (both iterative and recursive)



Factorial Formula

$$n! = n \times (n-1) \times (n-2) \times ... \times 1$$

$$1! = 1$$
$$2! = 2 \times 1 = 2$$
$$3! = 3 \times 2 \times 1 = 6$$
$$4! = 4 \times 3 \times 2 \times 1 = 24$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Lets review some code samples

- Fibonachi: 1, 1, 2, 3, 5, 8, 13, 21 ….

  f(n) = f(n-1) + f(n-2) f(0) = 1, f(1) = 1

The Golden Ratio

$$\varphi = 1.618034...$$

- The golden ratio is achieved already in the 11th number:

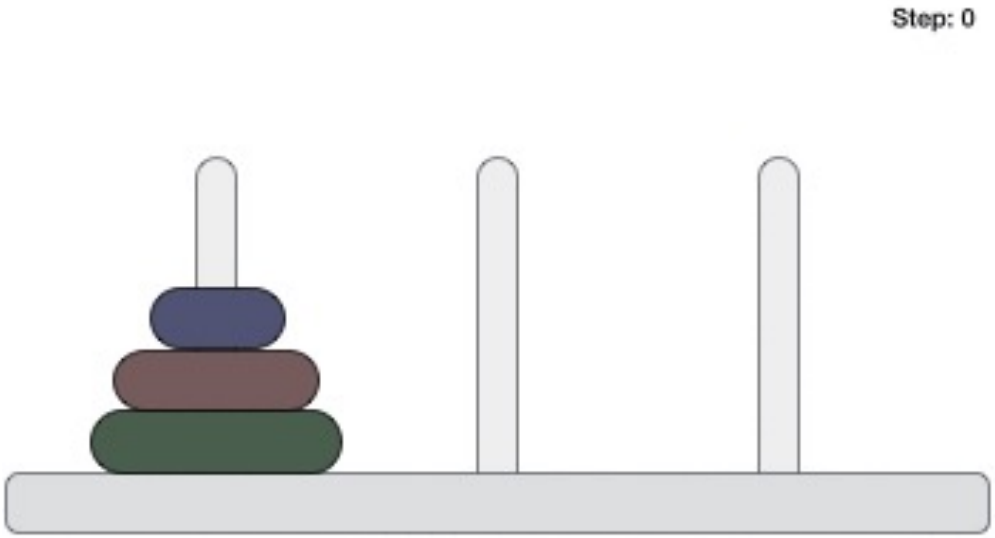
$$\frac{F_{11}}{F_{10}} = \frac{89}{55} = 1.6181818...$$

- And gets more and more accurate:

$$\frac{F_{16}}{F_{15}} = \frac{987}{610} = 1.61803...$$

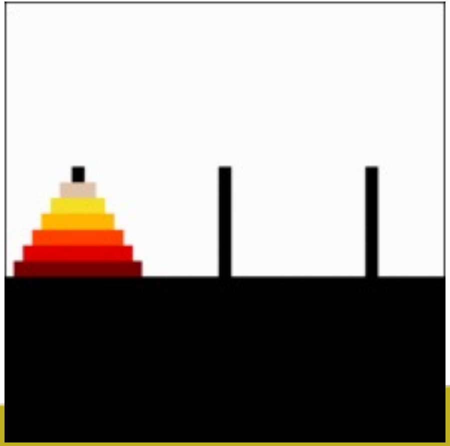
Lets review some code samples

There is a famous recursion called Hanoi Towers

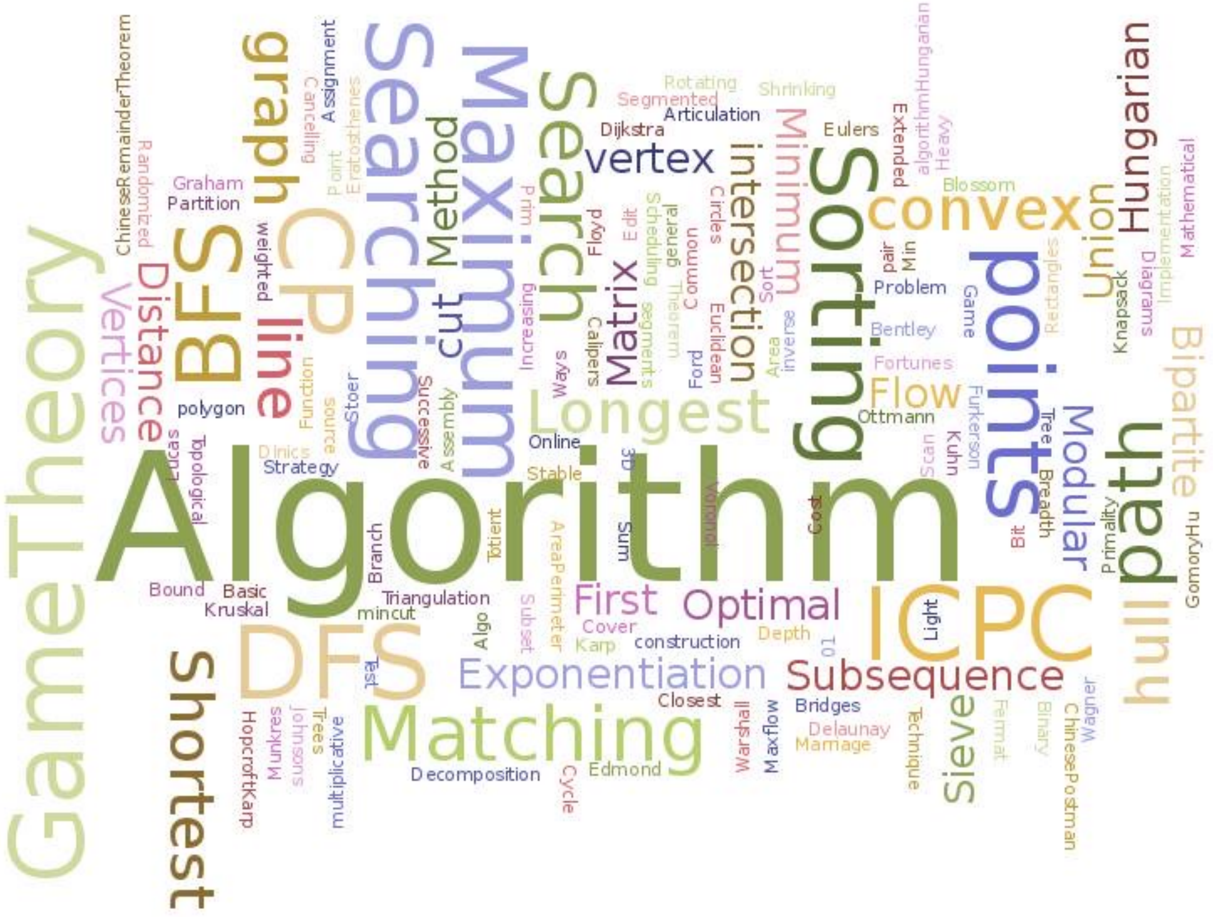


```
function hanoi(n, from, to , via){
  if (n==0) return;
  hanoi(n-1, from, via , to);
  moveDisk(from,to);
  hanoi(n-1, via, to , from);
}
```

here is a simulator



Introduction to Complexity



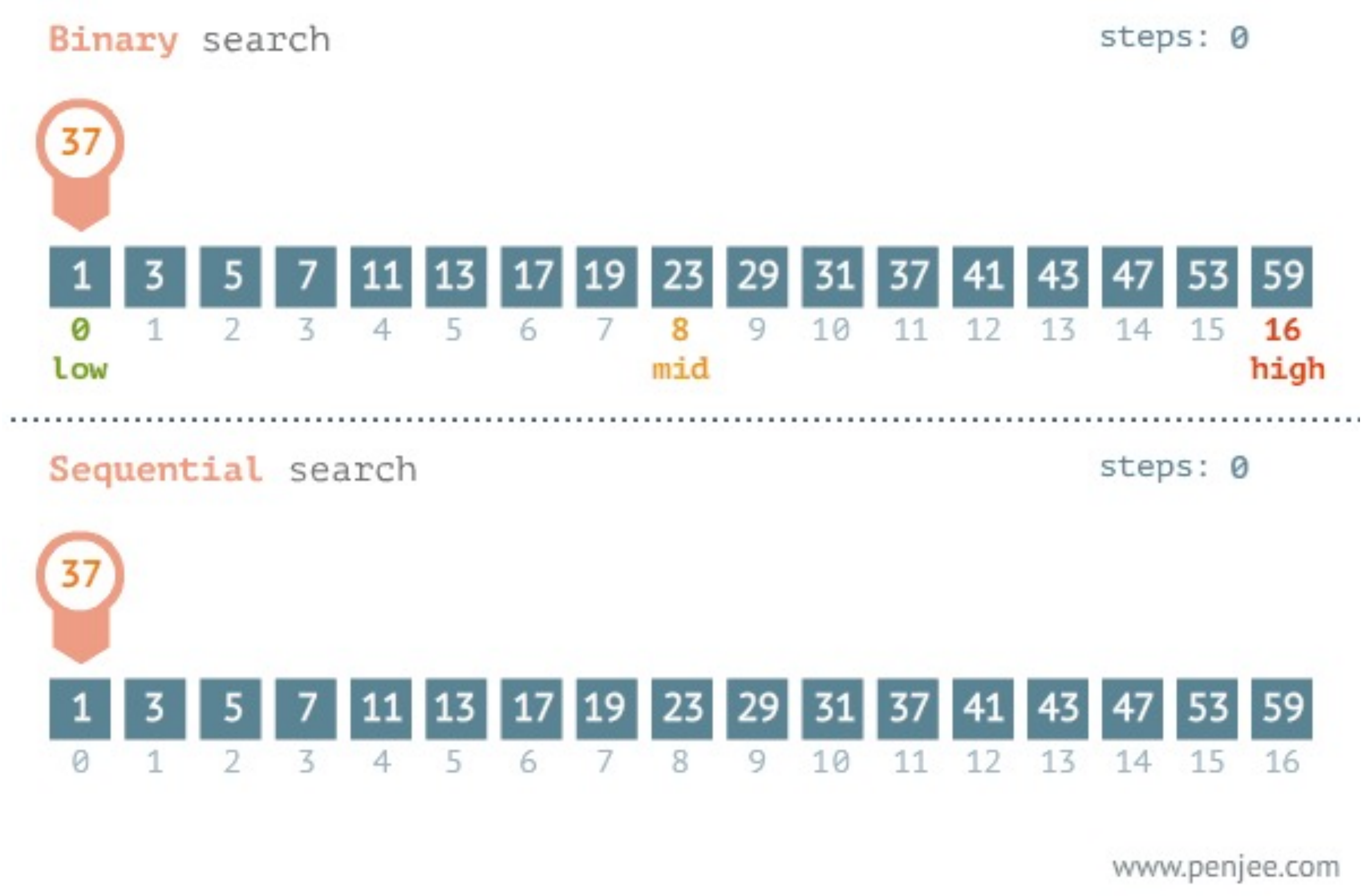
Lets Discuss Complexity

Also known as – The Big O Notation

- 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*
  - Complexity: $O(Log_2N)$ $Log_2 1024 = 10$
  - Its usually implemented with recursion
  - Lets review the binarySearch function

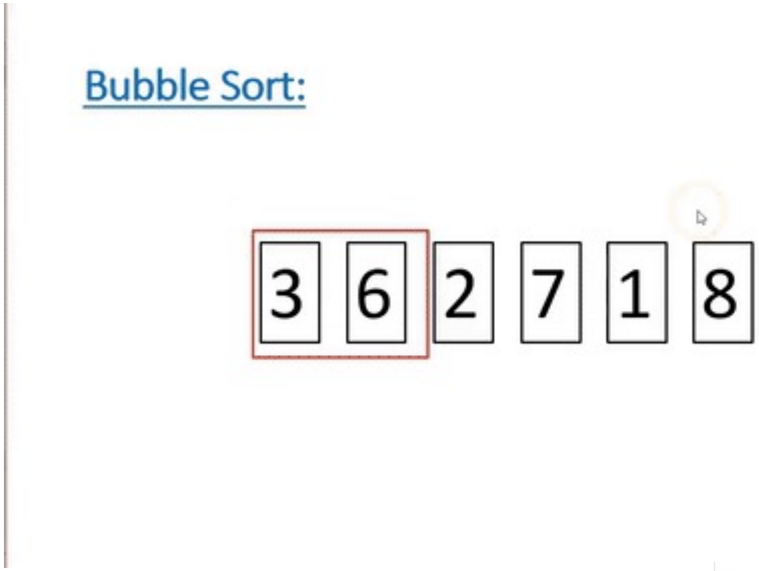
binarySearch

- Finding in an array normally cost O(N) (lets say 1000)
- But if array is sorted we can find much quicker: O(Log_2N) (Only 10 compares)



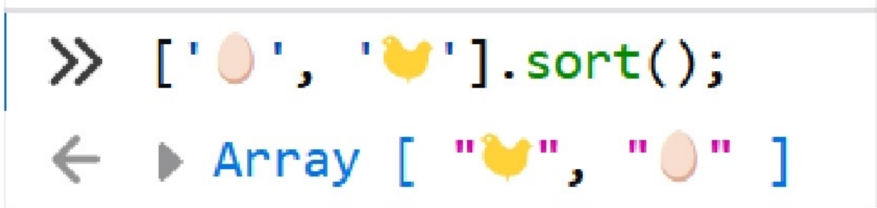
Let's Discuss Sorting

- Introducing Bubble sort
  - The Idea
  - Lets review the implementation
  - Complexity: O(N^2)



Bubble Sort:

3	6	2	7	1	8



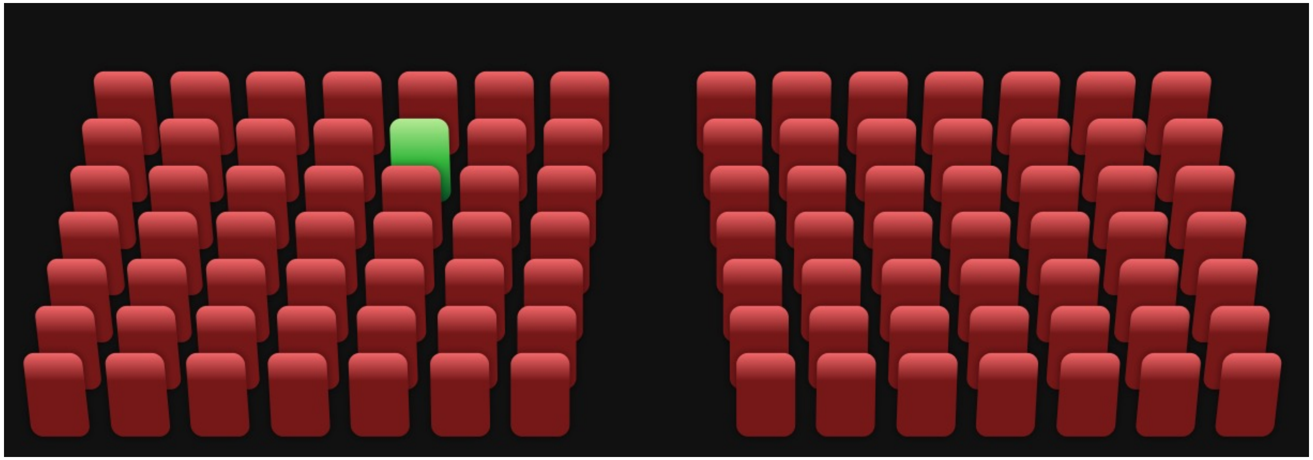
```
» ['🥚', '🐥'].sort();
← ▸ Array [ "🐥", "🥚" ]
```

Let's Discuss Sorting

- There are many sorting algorithms
- There are at best O(N * Log$_2$N)
- N === 1000   => O === 10,000

- Some of them are implemented using recursions
  - Lets overview the merge-sort algorithm

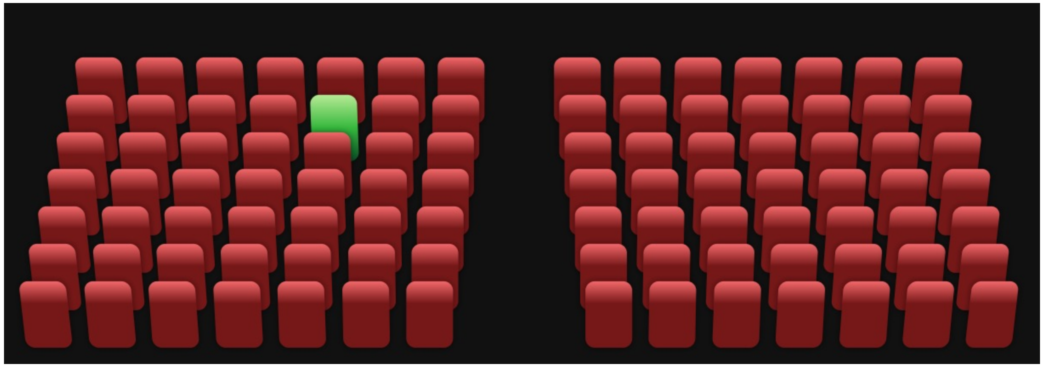
Let's Plan Seat Booking in a Cinema



Let’s identify global data names and structures

Let's Plan Seat Booking in a Cinema

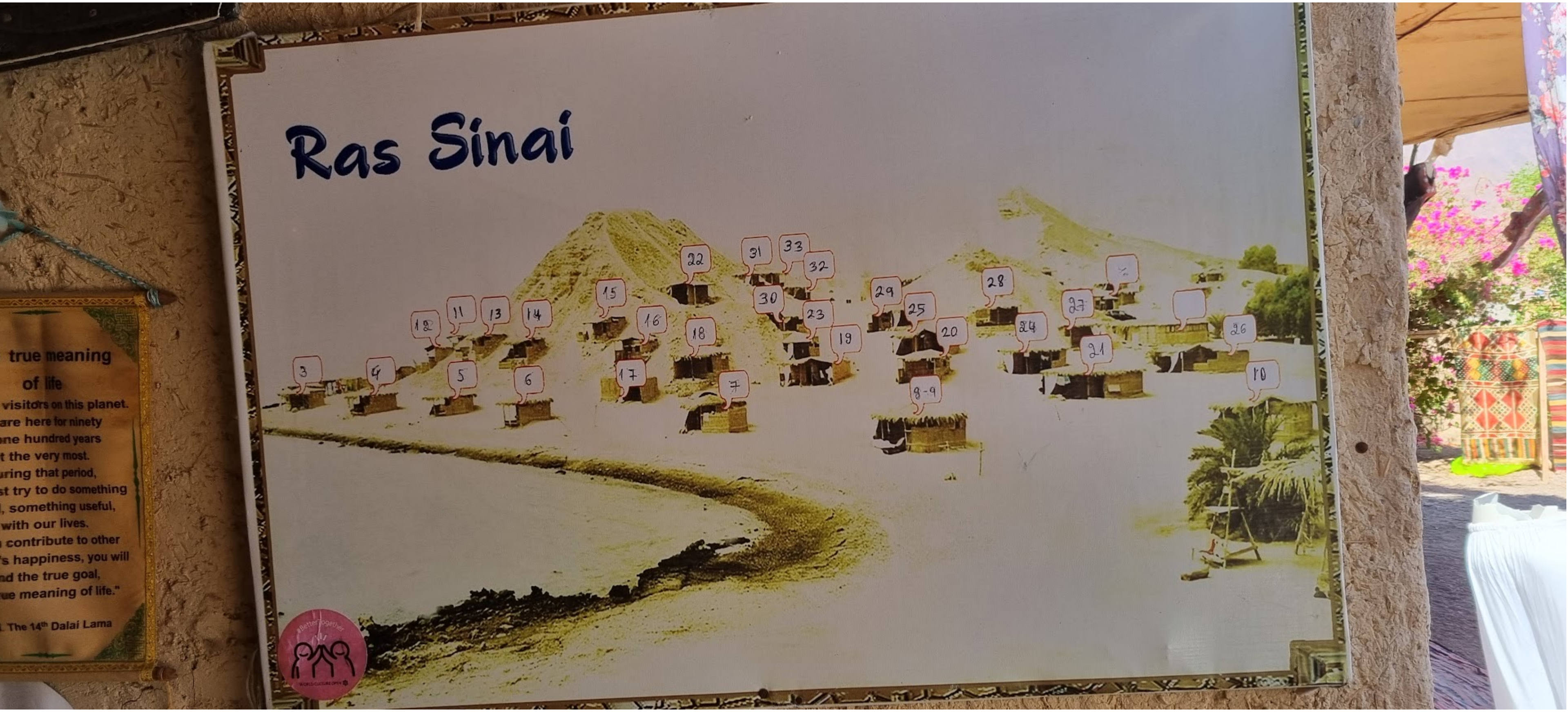
- Identify global data structures:
  - gCinema
  - gElSelectedSeat



```
var gElSelectedSeat = null;
var gCinema = createCinema();
renderCinema();


function createCinema() {}
function renderCinema() {}
function cellClicked(elSeat, i, j) { }
function showSeatDetails(pos) {}
function hideSeatDetails() { }
function bookSeat(elBtn) { }
```

Is our solution reusable?

Could we reuse our solution for solving more business cases?

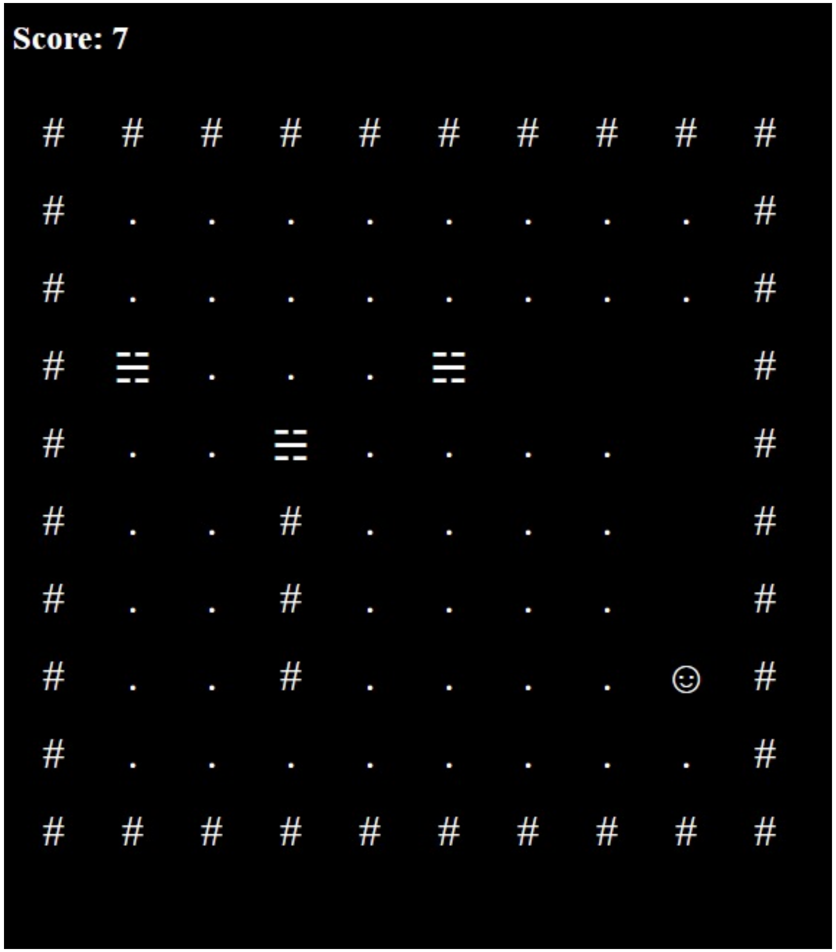


Modern Coding

- Coders work closely with the internet to find answers and move forward quicker
- Specifically, as web developers we will usually find our answers in 2 places:
  - MDN – The largest most updated source for web development documentation
  - Stack Overflow – a platform where developers ask / answer each other
- Lets have a look together

Lets Plan Pacman

- Identify global data structures:
  - gBoard
  - gPacman
  - gGhosts



```
Score: 7

#  #  #  #  #  #  #  #  #  #
#  .  .  .  .  .  .  .  .  #
#  .  .  .  .  .  .  .  .  #
#  ☵  .  .  .  ☵           #
#  .  .  ☵  .  .  .  .     #
#  .  .  #  .  .  .  .     #
#  .  .  #  .  .  .  .     #
#  .  .  #  .  .  .  .  ☺  #
#  .  .  .  .  .  .  .  .  #
#  #  #  #  #  #  #  #  #  #
```

```
const WALL = '#';
const FOOD = '.';
const EMPTY = ' ';
```

```
pacman = {
    location: {
        i: 3,
        j: 5
    },
    isSuper: false
  };
```

```
ghost = {
     location: {
         i: 3,
         j: 3
     },
     currCellContent: FOOD
   };
```

Victorious!

You have successfully grasped the basics of

Javascript

as a programming language