★ Ridiculous $\xleftrightarrow[\text{range}]{\text{wide}}$ Sublime

★ Nice expressive power ⟶ broadest range of programmers

★ DOM ⟶ Document Object Model ⟶ cross-platform, language independent.

★ AJAX libraries ⟶ help solving the DOM problems.

★ JavaScript ⟶ fast, interpreted
DOM ⟶ slows it down.

★

$$\begin{tikzcd}
 & \text{Java Script} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\text{Self} \arrow[d] & \text{Java} \arrow[d] & \text{Scheme} \arrow[d] & \text{Perl} \arrow[d] \\
\begin{array}{l}\text{(a) Prototypal Inheritence.}\\ \text{(b) Dynamic Objects}\end{array} & \begin{array}{l}\text{(a) Syntax}\\ \text{(b) Conventions (X)}\end{array} & \begin{array}{l}\text{(a) Lambda functions.}\\ \text{(b) Loose typing}\end{array} & \begin{array}{l}\text{(a) Regular Expressions (X)}\end{array}
\end{tikzcd}$$

★ Bad Parts:

(a) Global Variables ⟶ No linker ; during compilation, all units get tossed in a common global namespace where all variable names can collide & interfere with each other.

(b) + ⟶ used to both "add" and " concatenate"

(..... contd.)

common source of error as it was inherited from Java, but is loosely typed unlike Java.

(c) Semicolon insertion $\longrightarrow$ Automatically done; when compiler gets an error, it checks for linefeed, turns it into a semicolon and then tries again.

```
return ;              return {
  auto insertion
{   //  ✗               //        ✓
}                     };
```

(d) typeof $\longrightarrow$ typeof (array) : object (not helpful)
typeof (Null) : object (wrong)

(e) with $\longrightarrow$ "avoid" : makes it slow
eval $\longrightarrow$ most misused $\longrightarrow$ "avoid".

(f) Phony arrays $\longrightarrow$ NO normal arrays;
they are hashtables : NO dimension

(g) == and != : Always use ===

(h) false :

Null :

Undefined: If return not specified, return undefined!

NAN : Has no value; returned for an operation that cannot produce normal result.

Infinity : 1.79769313.... e 308

Too many bottom values!

★ Objects can inherit ~~[illegible]~~ from objects → no class inheritance.

★ Functions can be members of objects.
↳ methods creation

★

<u>for.. in</u> → Statement that mixes inherited functions with the desired data members
→ NO use, common source of errors.

★ Blockless statements → add curly braces.

★ Expression statements → foo;

★ Floating Point Arithmetic → 0.1 + 0.2 !== 0.3
↳ Everything is floating point type.

★ ++ and -- → can be tricky code.

★ Switch → from FORTRAN goto
↳ move from one case to the next.
↳ falling through a switch statement.

★ Good Parts :

(a) Lambda $\longrightarrow$ Scheme : which came out of Carl Hewitt's work on Actor model.

Best Part : Powerful, Safe, Smart, good , flexible.

(b) Dynamic Objects $\longrightarrow$ take any object, at any time ; and add a new property to it or remove a property from it.

No need to go to a class, make a derived class to have an object just slightly different than the existing one.

(c) Loose Typing $\longrightarrow$ Good.

(d) Object literals $\longrightarrow$ Very nice notation for describing objects.

Inspiration for JSON data interchange format.

(e) Multiple implementation compatibility $\uparrow$

(f) JavaScript application can reach a potential audience of billions.

(g) Ideas of functions.

Inheritance

Inheritance is Object-oriented code reuse

(a) Classical ⟶ C++, JAVA.

(b) Prototypal ⟶ JavaScript.

(b) ⟶ (a) ✓ amazingly powerful.

(a) ⟶ (b) ✗

PROTOTYPAL INHERITANCE :

(a) Class free.

(b) Objects inherit from Objects

(c) Delegation or Differential inheritance: An object contains a link to another object.
So new-object contains only what different it contains from the old-object.
So, Objects are small! Saves object initializaⁿ time and memory consumption.

(d) New operator :- Necessary when calling a Constructor function.

- If omitted, global object is clobbered by the con
- No compile-time or run-time warning.

☆ Avoiding "Global" variables:

(a) **Slow method:** Define the variable within the function itself.

But slow, as it always re-initializes the variable.

(b) **Closure use:** A (return) function within a function.

So the inner function enjoys full access to the outer function even when outer not called.

(c) **A Module Pattern:** Outer ~~class~~ function has private shared access i.e. no global!

The inner (return) function returns/does useful stuff with access to the outer function

$\downarrow$

Powerful Constructor Pattern

Power Constructors

1. Make an object.
   (a) Object literal   (c) Object. create
   (b) new operator   (d) call another power constr.

2. Define some variables & functions.
   - These are the private members of the Object.

3. Augment the object with privileged methods
   - These methods have access to step (2).

4. Return the object.

Closure

A function object contains:
(a) A function (name, parameters, body) → inner function
(b) A reference to the environment (content)
↳ outer function

JSLint (use mainly)

Check if your code contains good parts only!

Query:

(a) 35.

(b) 38.

(c) 40 $\longrightarrow$ Module.

(d) "pop" implementation.

★ Why JavaScript?

(a) Language of the web-browser

(b) Don't need too much $\overset{\text{prior}}{_\wedge}$ programming knowledge, nor prior knowledge of JavaScript.

★ All numbers → Floating Point : 64-bit

All strings → "Characters" : 16-bit

NO char → create string with one character

NO int → Math.floor (number).

Strings are immutable!

length → string.length

★ Closure → Source of enormous expressive power.

Function object created by a function literal contains a link to the outer context.

★ Public methods : Methods which get their object context using "this" are called "Public methods".

* Test for arrayness:

```
var is_array = function (value) {
    return value &&
           typeof value === "object" &&
           type of value. length === "number" &&
           type of value. splice === "function" &&
           !(~~[illegible]~~ value . propertyIsEnumerable("length"));
};
```

* Regular Expressions:

^ : string start $: end of url → anchor the reg expr".

(a) Scheme : letter (A-Za-z+) → optional

(b) / : slash (0,1,2 or 3) (\/{0,3})

(c) host : letter or digit (0-9.\-A-Za-z+)

(d) port : digit (\d +) → optional

(e) path : any character but not ? or # ([^$#]*) → optional

(f) Query : —"— # ([^#]*) → Optional

(g) Hash : —"— ~~[illegible]~~ end line ([^.]*) → optional

start ? : → non-capturing group.

end ? → optional.

(...) → capturing group.

✯ Array methods :

(a) Push $\longrightarrow$ add element at end of array.
(b) Pop $\longrightarrow$ return element last in the array.
(c) unshift $\longrightarrow$ add element at start of array.
(d) shift $\longrightarrow$ pop element first in the array.
(e) Concat $\longrightarrow$ add b to a at the end a.concat(b).
(f) Join $\longrightarrow$ a.join(" ") : default ",".
(g) Slice $\longrightarrow$ a.slice(start, end)
(h) Splice $\longrightarrow$ a.splice(start, no.of ele.delete)
(i) reverse $\longrightarrow$ reverse the array.
(j) Sort $\longrightarrow$ always sorts as "String".
To incorporate int; create your own function.

✯ Function methods :

(a) Apply $\longrightarrow$ function.apply(object, array of arguments)

$$\begin{tikzcd} \text{object} \arrow[d] \\ \text{bound to "this"} \end{tikzcd}$$

NOTE : [ ].slice.call(arguments, start)
$\hookrightarrow$ returns an array of arguments from arg[start ; end]

✮ Number methods :-

(a) toExponential $\longrightarrow$ converts number to a string in exponential form

Eg: Math.PI.toExponential(2)

3.14e+0

(b) toFixed $\longrightarrow$ string in decimal form. (default=0)

(c) toPrecision $\longrightarrow$ —"— (between 1 and 21)

(d) toString $\longrightarrow$ num to string (parameter: radix)

(default: 10) $\longleftarrow$ (2, 10, 8, 16, etc.)

✮ Object methods :

(a) Object.hasOwnProperty(name) $\longrightarrow$ *not intended to Prototype chain*

$\hookrightarrow$ returns true <u>or</u> false

- if Object has property "name"
- if object doesn't have property "name".

Eg:

```
var  a =  { member : true };
var  b =  Object.Create(a);
var  u =  b.hasOwnProperty("member"); // u → true
var  v =  b.member ; // v → false
```

★ <u>String methods :-</u>

(a) string. charAt (pos) $\xrightarrow{\text{yes}}$ returns "string" (no char in JS)

$\quad\quad\quad\quad\quad\quad\quad\quad\quad$ $\xrightarrow{\text{no}}$ returns "empty string" (if pos < 0 or pos > length)

(b) String. CharCodeAt (pos) $\xrightarrow{\text{yes}}$ returns integer representation of <u>code point value of the character</u> at pos.

$\quad\quad\quad\quad\quad\quad\quad\quad\quad$ $\xrightarrow{\text{no}}$ returns NaN.

(c) String. concat (string..) ⟶ generally use "+"

(d) String. indexOf (searchString, position)

searchString ⟶ string to search

position ⟶ optional "start point"

↓

if(yes): returns position of 1ST matched char.
else : returns -1

(e) string. lastIndexOf (searchString, position)

↓

same as (d) except it starts from the end of the string.

(f) string. localCompare (that) ⟶ compares 2 strings
↳ return 0 if equal
↳ return -1 if string < that

similar to array. sort comparison function.

(g) string. replace (search Value, replace Value)
↳ replace ONLY the 1st occurrence of the "search Value" with "replace Value".
↳ regular expression ⟶ with 'g' tag; replace all
↳ without 'g' tag, replace 1st occ.

(h) string. search (regexp) ⟶ same as indexOf except argument = regexp & != string.

(i) String. slice (start, end) ⟶ end : default ⟶ length.
↳ if -ve → add length.

goes from start to (end - 1).

(j) String. Split (separator, limit) → optional, limit the no. of pieces.
↳ string or regexp

(k) string. substring(start, end) $\longrightarrow$ same as slice
↳ can't handle -ve.

(l) string.toLocaleLowerCase()
(m) string.toLocaleUpperCase() } for rules of locale.

(n) string. toLowerCase()
(o) string. toUpperCase() } intuitive.

(p) string. fromCharCode (char....)
↳ var a = string.fromCharCode (67,97,116);

// a = Cat.

Angular JS

PHONECAT TUTORIAL

★ Unit tests :

a) To check if/ensure that the JS code in our app" is operating correctly. Tests small isolated parts of the application.

b) kept in the test/unit directory.

c) opens Chrome browser & connects to Karma.

d) always running in the background.

★ End-to-End tests :

a) Ensure that applica" as a whole operates as expected.

b) test client-side application i.e. if views are displaying and behaving correctly.

c) Using stimulating real user interaction with real applica" running in browser.

★ BOOTSTRAPPING : (Angular initialization process)

(a) < html lang = "en" ng-app > auto-bootstrap your application

ng-app : Angular directive

- markers of DOM elements (attributes, ele. name, comment, css class)
- tell HTML compiler ($compile : Angular JS's compiler) to attach specified behavior to that DOM element or even transform the DOM element and its children.

(a) flag the html element that Angular should consider to be the root element of our application.

(b) Angular JS script tag:

```
<script src="bower-components/angular/angular.js">
```

- Download angular.js script and register a callback that executes when HTML is fully downloaded.
- Angular looks for ngApp directive.
- If found, it will bootstrap the applica" to the root of app" DOM → which is where ngApp is defined.

★ Important features of Angular's templating capabilities :-

1> $\boxed{\text{Binding}}$ ⟶ denoted by {{ }}

2> $\boxed{\text{Simple expression}}$ ⟶ used in this binding.

→ It tells Angular that it should evaluate an expression and replace/insert the result into the DOM in place of the binding.

→ **Advantage:** instead of a one-time insert, we get efficient conti. updates whenever the result of the expression evaluation changes.

→ (a) Content ⟶ scope ✓ global ✗.
(b) null and Undefined.
(c) No loops, conditional, exceptions.
(d) No funcⁿ declarations.
(e) Can use filters.

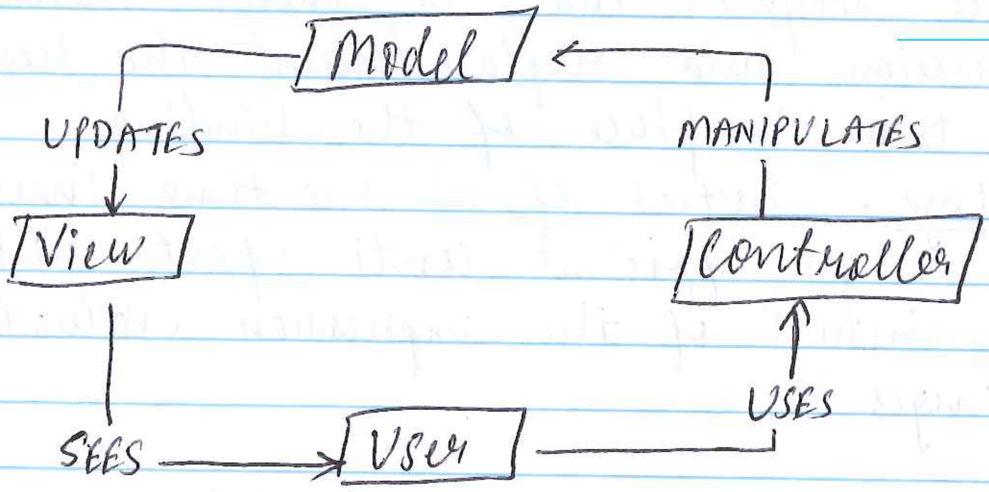
{{:: expression}} ⟶ One-time binding.

* STATIC TEMPLATE :-

Normal HTML lists

* ANGULAR TEMPLATES :-

(a) Model - view - controller (MVC) design pattern



(b) "View" is a projection of the model through the HTML template.
So, change in model ⟶ refresh binding points ⟶ update View.

(c) "Scope" ⟶ allows template, model and controller to work together.

$$\boxed{\text{changes in model} \rightleftarrows \text{changes in view}}$$

★ Filtering Repeaters :- (Full text search)

(a) Data Binding : when page loads, Angular binds the name of the input box to a variable of the same name in the data model, and keeps the two in sync

★ 2-way Data Binding :- (User control the order of items)

(a) Dynamic ordering using new model property creation wiring it with repeater, and letting data binding magic take control.

★ XHRs and Dependency Injection :- ($http)

(a) $http : make a http request to your Web Server to fetch the data in the .json file

(b) .json : data-interchange format [key : value pairs]

(c) DI : Angular services are managed by the DI. It helps make our apps -

well structured ⟶ separate components for presentation, data and control.

loosely coupled ⟶ component dependencies resolved by DI and not component themselves

(d) $http makes an HTTP GET request to our web server, asking for .json
The server responds by providing the data in the .json file.

→ $http service returns an object with a success method.
→ We call this method to handle the asynchronous response and assign the data to the scope of the controller.

(e) Minification: Process of removing all unnecessary characters from the source code without changing its functionality. Unnecessary characters are:
- white space characters.
- new line characters.
- comments
- Block delimiters.

} add readability to the code but are not required for execution!

Minification reduces the amount of data that needs to be transferred.

(f) XHR : XMLHttpRequest

API available to Web browser scripting languages to send HTTP or HTTPS request to a web server and load the server response data back into the script.

☆ Templating Links and Images :-

(a) Add thumbnail images for phones.
(b) Add links for phones (as of now go nowhere).

→ use double-curly brace binding in the "href" attribute values

→ use ngSrc directive which prevents the browser from using the Angular {{ exp }} as a markup literally & hence avoids initiating a request to invalid URL.

☆ Routing and Multiple Views :-

→ Create a layout template and build an app that has multiple views by adding routing, using an Angular module called "ngRoute".

→ index.html → layout template.
phone-list →
phone-details → } Partial templates.

Partial templates are included in the layout template depending on the "current route".

→ $routeProvider provides the $route [illegible] service.

→ $route service is usually used with the "ngView" directive
→ to include the view template of current route into the layout template.

→ You need to add modules as dependencies of our app!!!

→ Use the config() method to request the $routeProvider to be injected into config func^n & use the $routeProvider.when() method to define our routes.

→ $route service uses the route declaration "/phones/:phoneId" ⟹ as the template that is matched against the current URL.

All variables defined with the : notation are extracted into the $routeParams object.

★ <u>More Templating</u> :-

→ Add details to each phones : which would be displayed when the user clicks on a phone in the phone list.

→ Update the PhoneDetailCtrl controller with the use of $http service to fetch data from JSON files.

→ To construct URL for the HTTP Request, we use $routeParams.Id extracted from the current route by the $route service.

★ Customized filters :-

→ Create custom filters to convert text strings to glyphs.

✓ for "true" → \u 2713.
✗ for "false". → \u 2718.

→ Create a new module and register the custom filter with this module.

→ Syntax :- {{ expression | filter }}

★ Event Handlers :-

→ Add a clickable phone image swapper to the phone details page.

→ In phonedetails controller
↓

(a) mainImageURL → model property
└→ default image [0]

(b) event handler → set Image (ImageURL)
└→ mainImageURL = ImageURL

→ ngSrc = "{{ mainImageURL" and not just main—
→ ng-click = "setImage (img)"
↑
< img ngsrc = "{{ img}}" .... >

EGGLY INTRO

(Managing Bookmarks)

★ Module → Organizational mechanism to group functionality.

★ ng-show : toggle between

★ ng-controller: attach a controller class to the view.

★ ng-click: specify custom behavior when an element is clicked.

★ ng-submit: enable binding angular expressions to onsubmit events.

★ ng-model : binds an input, select to a property on the scope.

★ ng-init : evaluate an expression in the current scope.

★ ng-show : shows or hides the given HTML element.