JavaScript questions and terms

# Activation Object (AO)

* A special object created when a function is called (*activated*).
* It’s filled with formal parameters and an arguments object (an indexed map of parameters)
* The AO is then used as a ***variable object*** of the function context

\*\* see [www.dmitrysoshnikov.com/ecmascript/javascript-the-core/#activation-object](http://www.dmitrysoshnikov.com/ecmascript/javascript-the-core/#activation-object)

Actual Parameter

An actual value (argument) that is passed into a function. See “Formal Param”.

# Apply

Calls a function with a given “this” value, and arguments provided as an array

Ex/

// min/max number in an array

var numbers = [5, 6, 2, 3, 7];

// using Math.min/Math.max apply

var max = Math.max.apply(null, numbers);

# Arrow functions

The lexical “this”, which they inherit from the parent context.

# Callbacks

(<http://blog.millermedeiros.com/callbacks-promises-signals-and-events/> )

Used when you have an asynchronous operation that should notify the caller about its’ completion.

(<http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/> )

A Callback function (aks Callback Patter) is a higher order function, meaning it’s a func that is passed to another func.

Callbacks are great for cases where you have an action that triggers a direct reaction (eg. animation, ajax, batch processing).

When we pass a callback function as an argument to another function, the callback is executed at some point inside the containing function’s body just as if the callback were defined in the containing function. This means the callback is a closure.

# Closure

* A combination of a code block and all statically/lexically saved parent scopes.
  + See <http://dmitrysoshnikov.com/ecmascript/javascript-the-core/#closures>
* Functions that are free/independent vars which are used locally, but defined in an enclosing scope. These functions remember the environment in which they were created.
* a function, and the environment in which that function was created (see <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures> )

# Equality Comparison

## “===” Strict equality

* Both sides must be the same type, otherwise they are considered unequal
* Neither value is implicitly converted

var num = 0;

var obj = new String("0");

var str = "0";

var b = false;

console.log(num === num); // true

console.log(obj === obj); // true

console.log(str === str); // true

console.log(num === obj); // false

console.log(num === str); // false

console.log(obj === str); // false

console.log(null === undefined); // false

console.log(obj === null); // false

console.log(obj === undefined); // false

## “==” Loose Equality

* Compares two values AFTER converting one or both sides to a common type
* The ***final equality*** comparison is performed exactly as **===** performs it

# Formal Parameter

The identifier used in a method to stand for the value that is passed into the method by a caller. See “actual param”.

Ex/ processDeposit(*amount*)

# Function Definition (FD)

# Function Expression (FE)

* The function keyword can be used to define a function inside an expression.

Ex/ var x = function(y) {

return y \* y;

};

* Similar to a Function Statement, except that the function name can be omitted in a FE.
* FEs are not hoisted, as opposed to standard Functions which are hoisted.

# Function Statement

* A standard function declaration with specified parameters.
* Standard function declarations are hoisted, so they can be accessed prior to being declared
* Function Expressions are NOT hoisted

# IIFE (Immediately Invoked Function Expression)

* provides an easy way to create privacy
* vars inside can only be accessed inside the context

# Minification

* Reduces the file size by 30-80%
* Removes whitespace, newline chars, comments
* Sometimes removes block delimeters (used for code readability)

Reasons for minification

* Quicker user downloads
* Reduced bandwidth consumption of your website
* ***Combining*** js files reduces the number of http requests on your server

# Named vs. Anonymous Functions

# Promise vs. Event

See <https://www.joezimjs.com/javascript/javascript-asynchronous-architectures-events-vs-promises/>

## Events/Listeners

* Event Handlers (or Listeners) associate functions with DOM events.
* element.addEventListener('click', function() { /\* do stuff here\*/ }, false);

var p1 = new Promise();

## Promises

Represent an asynchronous operation that hasn’t completed yet

# Static (Lexical) Scope

* The scope of a variable is defined by its location in the source code
* Nested functions have access to variables declared in its outer scope

# Strict Mode

* It’s a restricted variant of JavaScript, which intentionally has different semantics from normal code (MDN website).
* Strict mode code and non-strict mode code can coexist, so scripts can opt into strict mode incrementally.
* It makes several changes:
  + Eliminates some silent errors by throwing the errors instead
  + Fixes mistakes in order to improve optimization
  + Prohibits some syntax likely to be defined in future versions of JS

Ex/ Variables must be declared first; makes it impossible to accidentally create a global var.

# This value

A this value is a special object which is related with the execution context, but NOT in the variable context. Therefore, it may be named as a context object.

# Variable Object (VO)

A *variable object (in abbreviated form — VO)* is a special object related with an execution context and which stores:

* variables (var, VariableDeclaration);
* function declarations (FunctionDeclaration, in abbreviated form FD);
* and function formal parameters

declared in the context.

\*\*\* in ES5 the concept of variable object is ***replaced with* lexical environments *model***