Front End Software Development

Introduction to JavaScript (weeks 1 - 6)

Week 03

Agenda

- Questions
- Arrays
- Functions
- Intermediate Array Methods
- Objects
- Equality

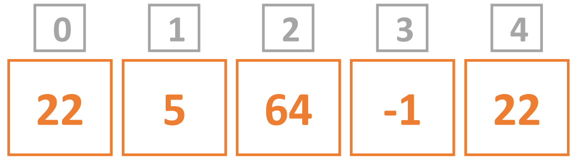
Questions?

You’re not alone if this week made you feel like this…

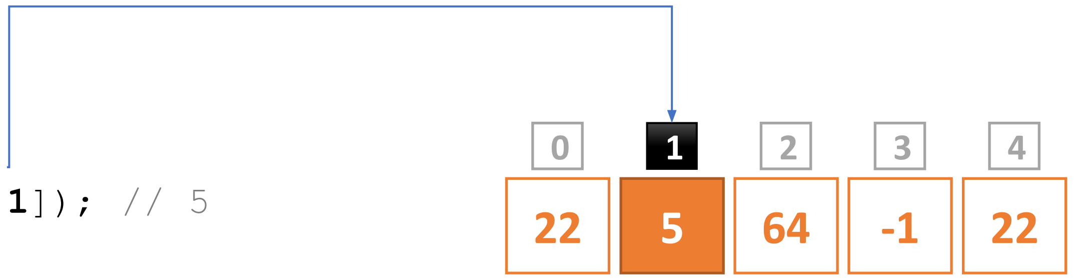
Arrays

- Used to store or group similar data together
- Recommended to pluralize variable name (add "s")
- 0 based index (0…N-1)

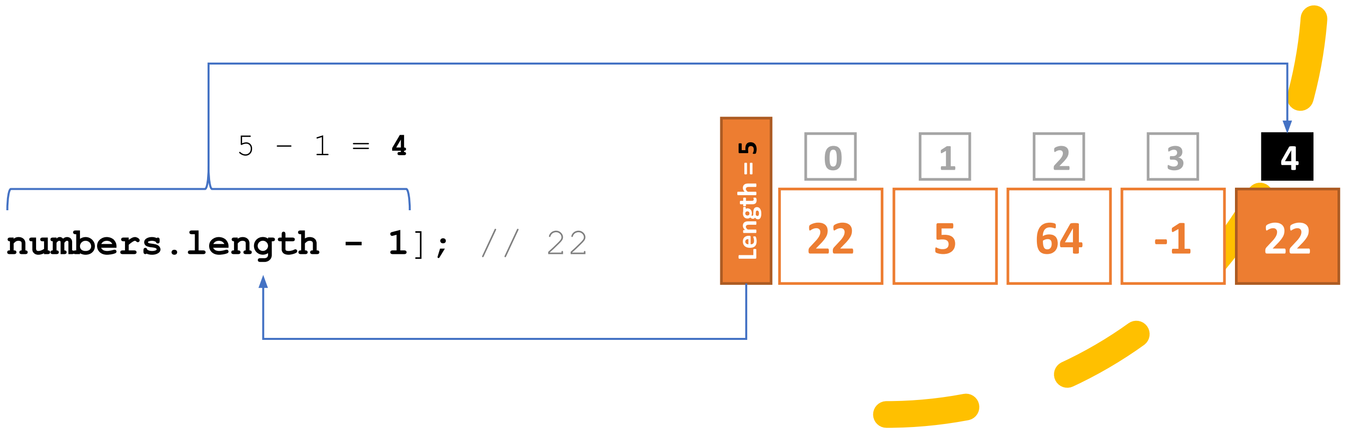
```
let numbers = [ 22, 5, 64, -1, 22 ];
numbers.length; // 5
```



```
console.log(numbers[1]); // 5
```



```
numbers[numbers.length - 1]; // 22
```



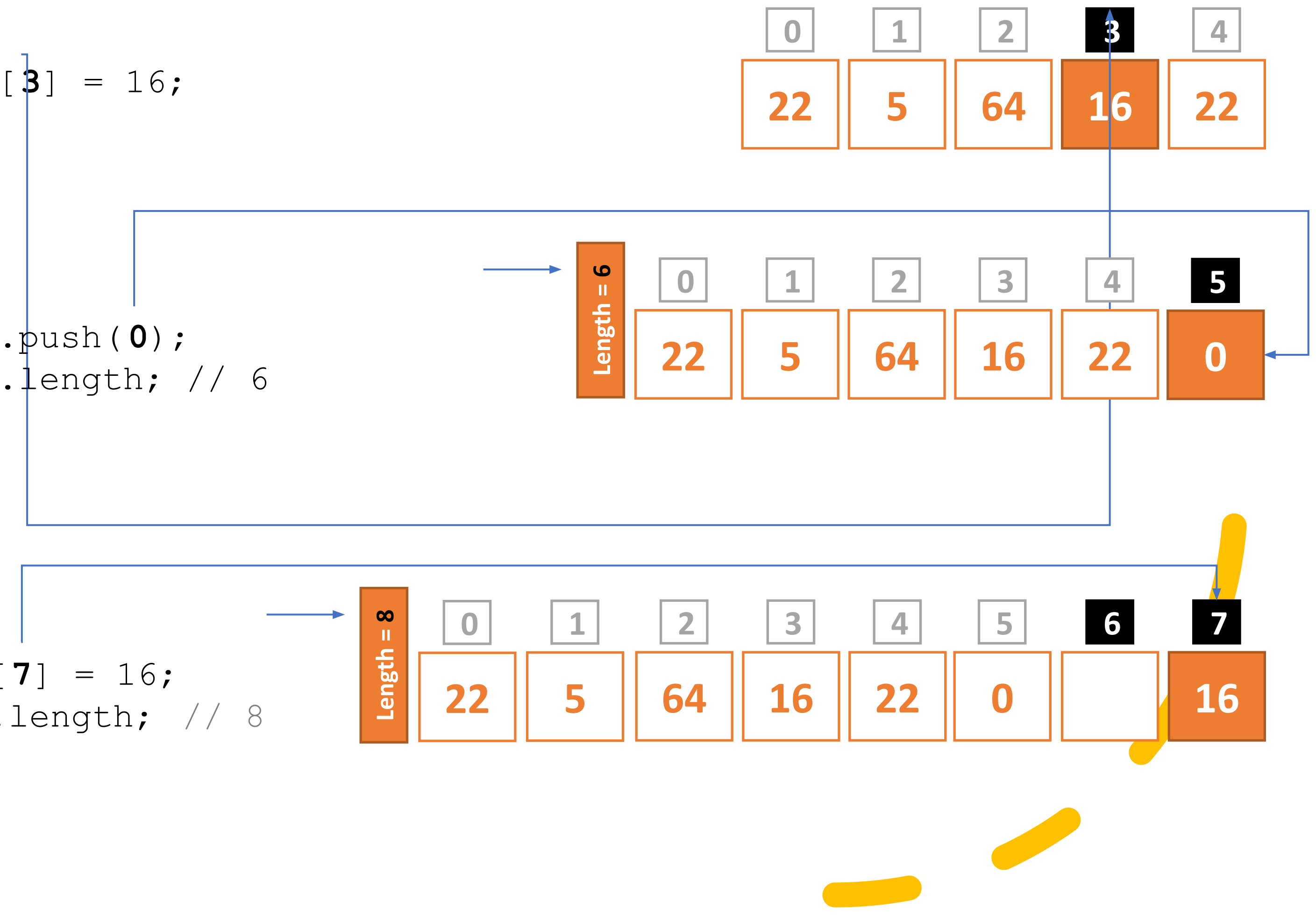
Arrays
(continued)

- Setting or Changing Values

```
numbers[3] = 16;
```

```
numbers.push(0);
numbers.length; // 6
```

```
numbers[7] = 16;
numbers.length; // 8
```

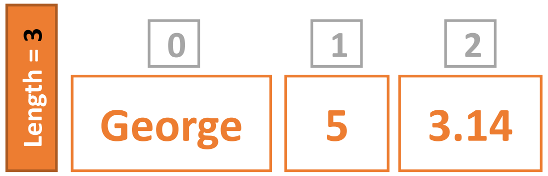


Arrays

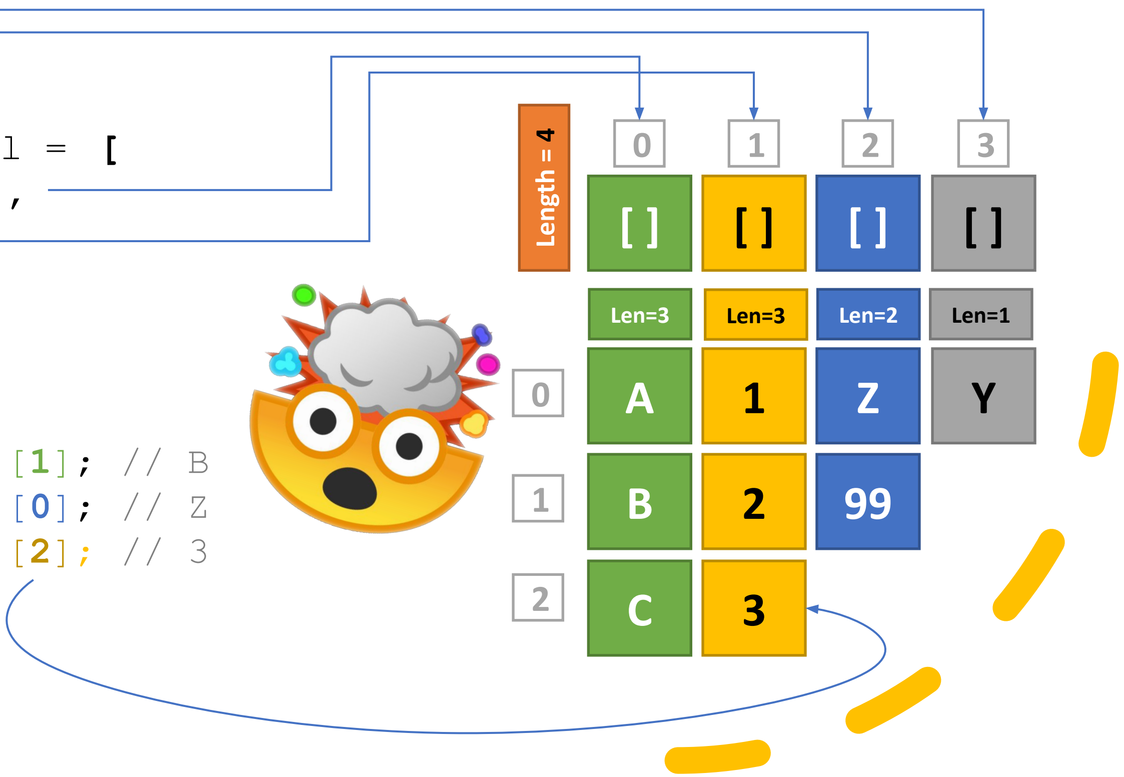
(continued)

- Arrays can hold mix / match different data types.

```
let stuff = [ "George", 5, 3.14 ];
```



```
let twodimensional = [
    [ "A","B","C" ],
    [ 1, 2, 3 ],
    [ "Z",99 ],
    [ "Y" ]
];

twodimensional[0][1]; // B
twodimensional[2][0]; // Z
twodimensional[1][2]; // 3
```

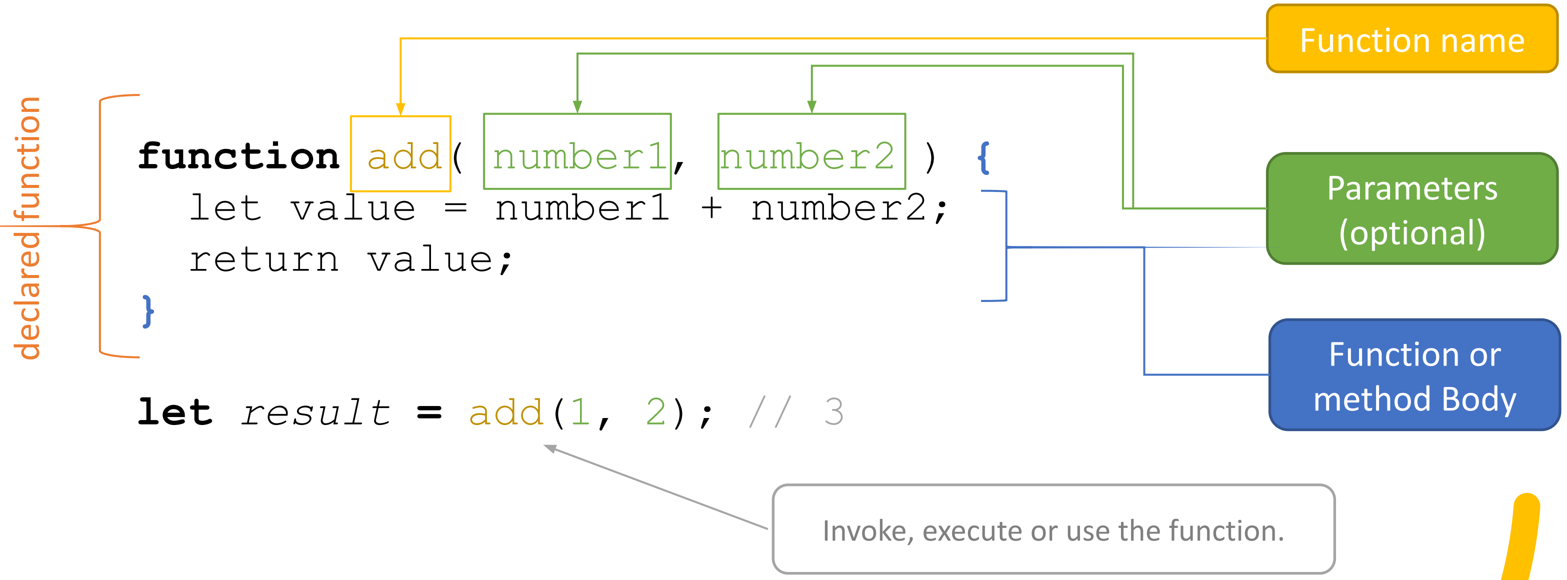


DEMO

Arrays: Initializing, Getting & Setting

Functions

- Allows “functionality” to be grouped or reused.
  - Standardized code, re-use Code
  - Use descriptive verbs (get, set, calculate, etc.)

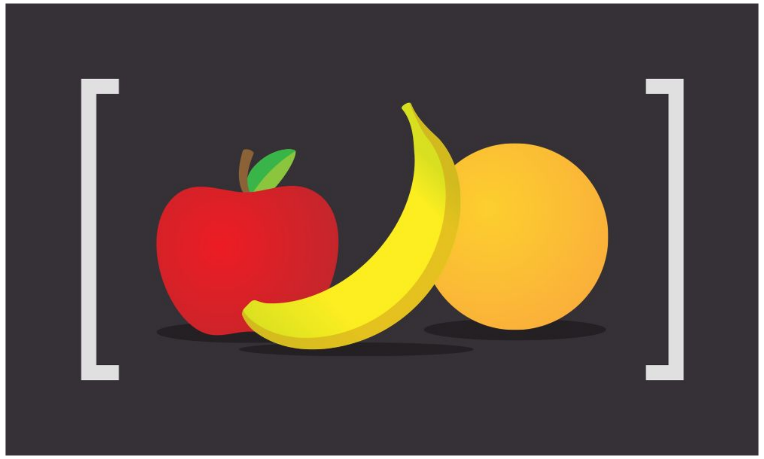


```
function add( number1, number2 ) {
  let value = number1 + number2;
  return value;
}

let result = add(1, 2); // 3
```

- *Optional* return value must be "captured" or assigned to a variable.

Intermediate Array Functions

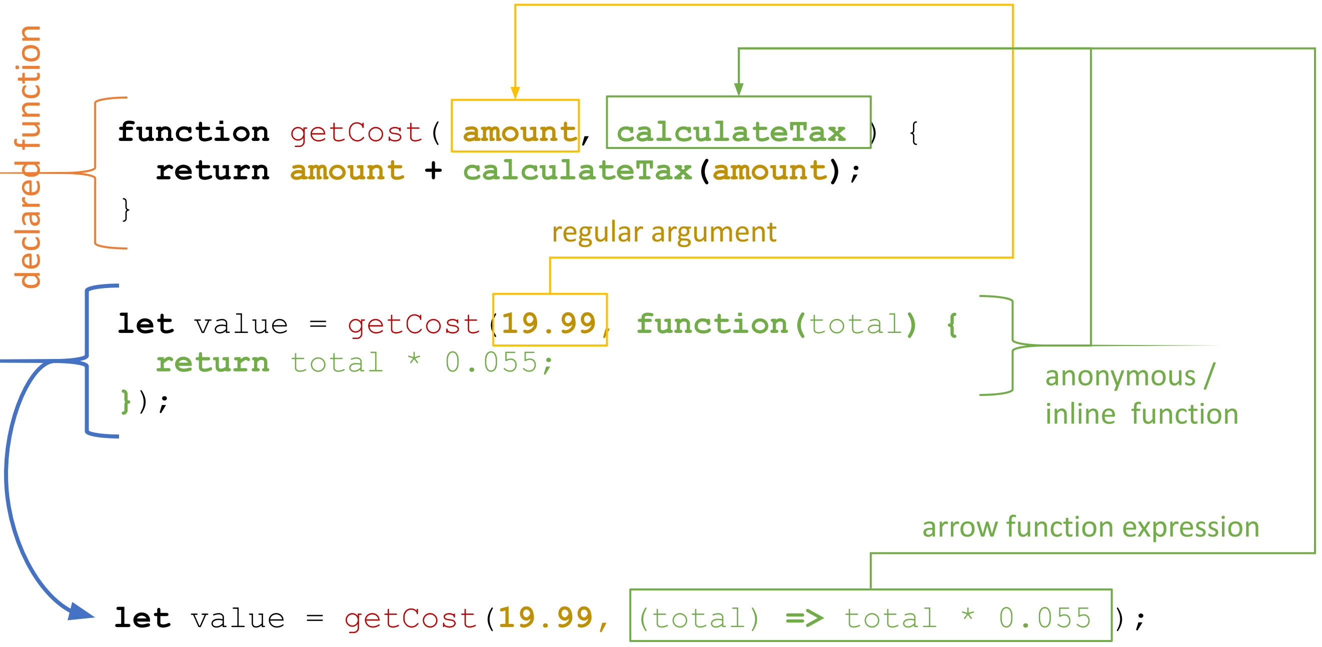
- Array Object
  - map
  - reduce
  - filter
  - forEach
  - splice



Intermediate Array Functions *(expressions)*

- In JavaScript, a function **IS** an object.

*“We can pass a function **into** a function as a parameter.”*



```
function getCost( amount, calculateTax ) {
  return amount + calculateTax(amount);
}
```

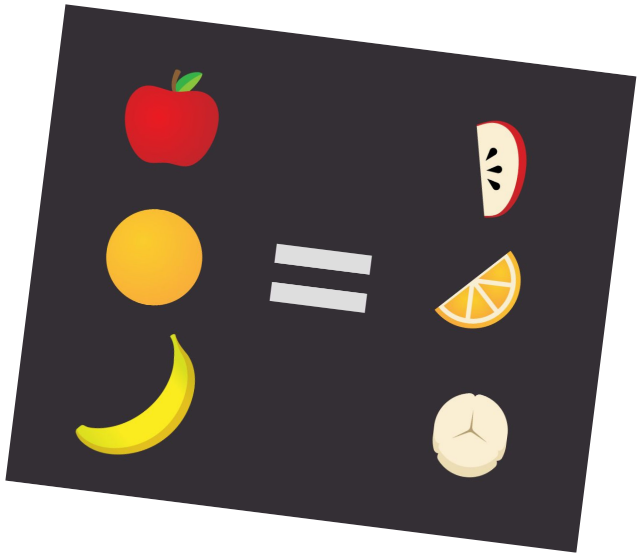
```
let value = getCost(19.99, function(total) {
  return total * 0.055;
});
```

```
let value = getCost(19.99, (total) => total * 0.055 );
```

Intermediate Array Functions *(map)*

- Array.prototype.map()

  The **map()** method creates a new array populated with the results of calling a provided function on every element in the calling array.



- Converts each item into a new item. The original array is **NOT** modified.

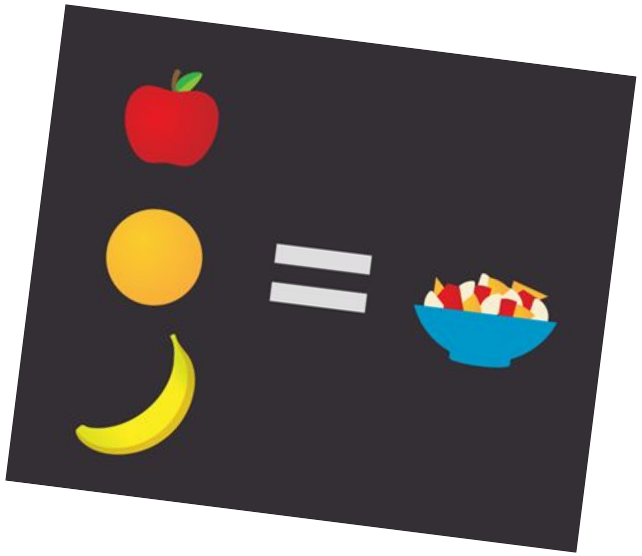
```
let fruits = [ "apple", "orange", "banana" ];

let slices = fruits.map(function(fruit) {
  return("slice of " + fruit );
});

fruits.length; // 3
slices.length; // 3
// [ "slice of apple", "slice of orange",
     "slice of banana" ]
```

anonymous / inline function

Intermediate Array Functions *(reduce)*

- Array.prototype.reduce()

  The **reduce()** method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element.



```
let fruits = [ "apple", "orange", "banana" ];

let bowl = fruits.reduce(function( previous, current ) {
  return(previous + " " + current);
});

console.log(bowl);
// apple orange banana
```

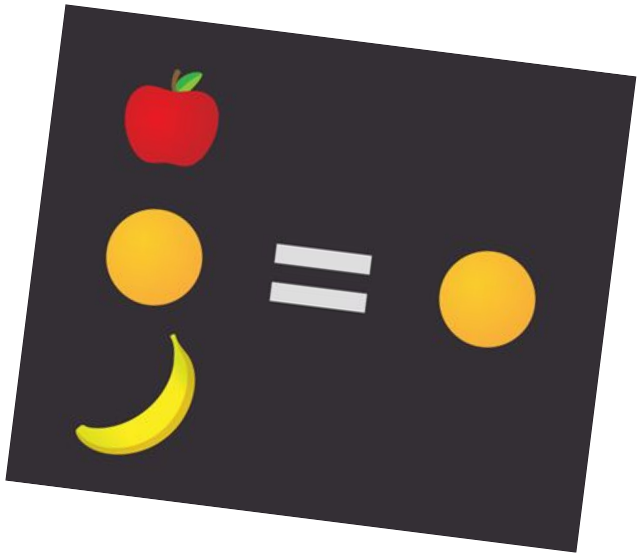
The previously returned value. Or if the first iteration, it contains the first item.

The current element.

Intermediate Array Functions *(reduce)*

- Array.prototype.filter()

  The **filter()** method creates a new array with all elements that pass the test implemented by the provided function.



The 0-based index is optional. But sometimes you'll need it!

```
let fruits = [ "apple", "orange", "banana" ];

let oranges = fruits.filter(function(fruit, index ) {

  return( fruit === "orange" );
});

oranges.length; // 1
```

Our **boolean** conditional. What are we looking for?

Remember, the filter will return an ARRAY of items that pass the conditional test (true). NOT just the first element found, but all elements that match. ***Note***: If none found an EMPTY array is returned NOT null!

Intermediate Array Functions *(forEach)*

- Array.prototype.forEach()

  The **forEach()** method executes a provided function once for each array element.

```
function eat(fruit) {
  console.log("Eating " + fruit + "... YUM!");
}

let fruits = [ "apple", "orange", "banana" ];

let oranges = fruits.forEach(function(fruit, index ) {
  console.log("["+index+"] " + fruit);
  eat(fruit);
});

// [0] apple
// Eating apple... YUM!
// [1] orange
// Eating orange... YUM!
// [2] banana
// Eating banana... YUM!
```

The 0-based index is optional. But sometimes you'll need it!

Intermediate Array Functions *(splice)*

- Array.prototype.splice()

  The **splice()** method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.
  *Note: Don't confuse **splice()** with **slice()**.*

```
let fruits = [ "apple", "orange", "banana" ];

let removed = fruits.splice( 1, 0, "lemon" );
console.log(removed); // []
console.log(fruits.length); // 4
console.log(fruits);
// ["apple","lemon","orange","banana"]
```

The 0-based index at which to start changing the array

An optional integer indicating the number of elements in the array to remove from start.

An array containing the deleted elements. ***Note***: If none are removed an EMPTY array is returned.

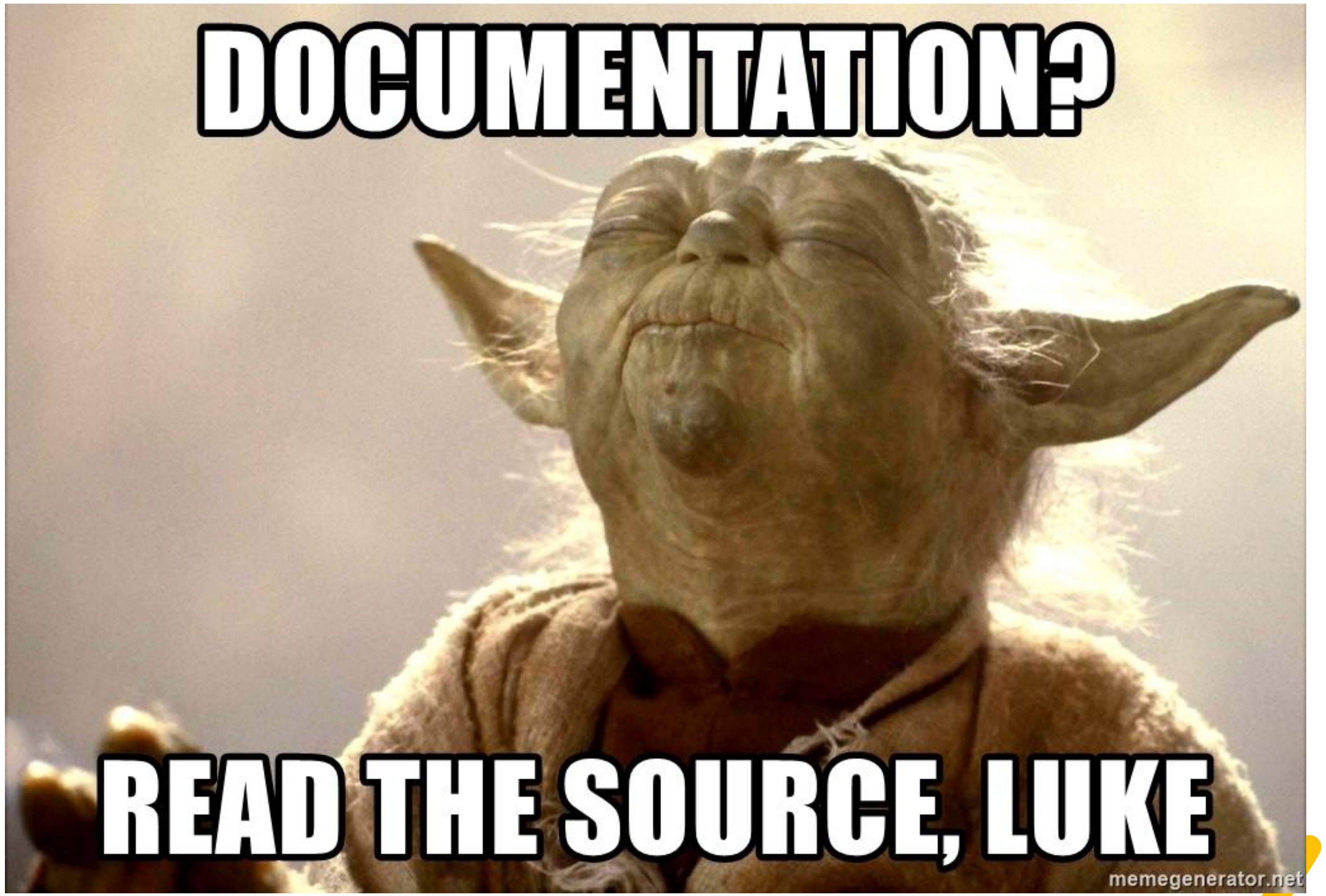
The elements to add to the array, beginning from start. If you do not specify any elements, splice() will only remove elements from the array.

```
fruits.splice(2, 2); // ["apple","lemon"]
fruits.splice(-1, 0, "grape", "peach", "cherry");
// ["apple","grape","peach","cherry","lemon"]
fruits.splice(0); // []
```

DEMO

Arrays: map, reduce, filter, forEach, splice

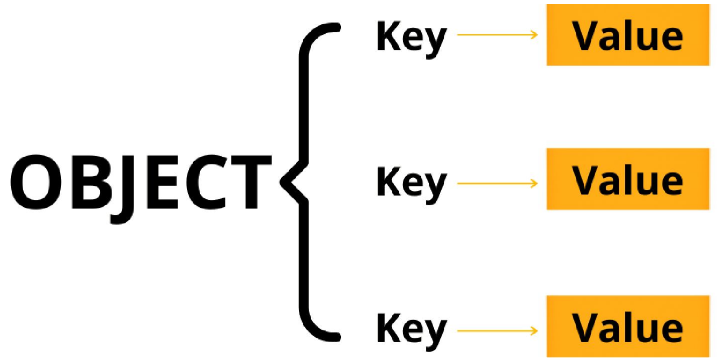
Intermediate Array Functions *(MORE!)*

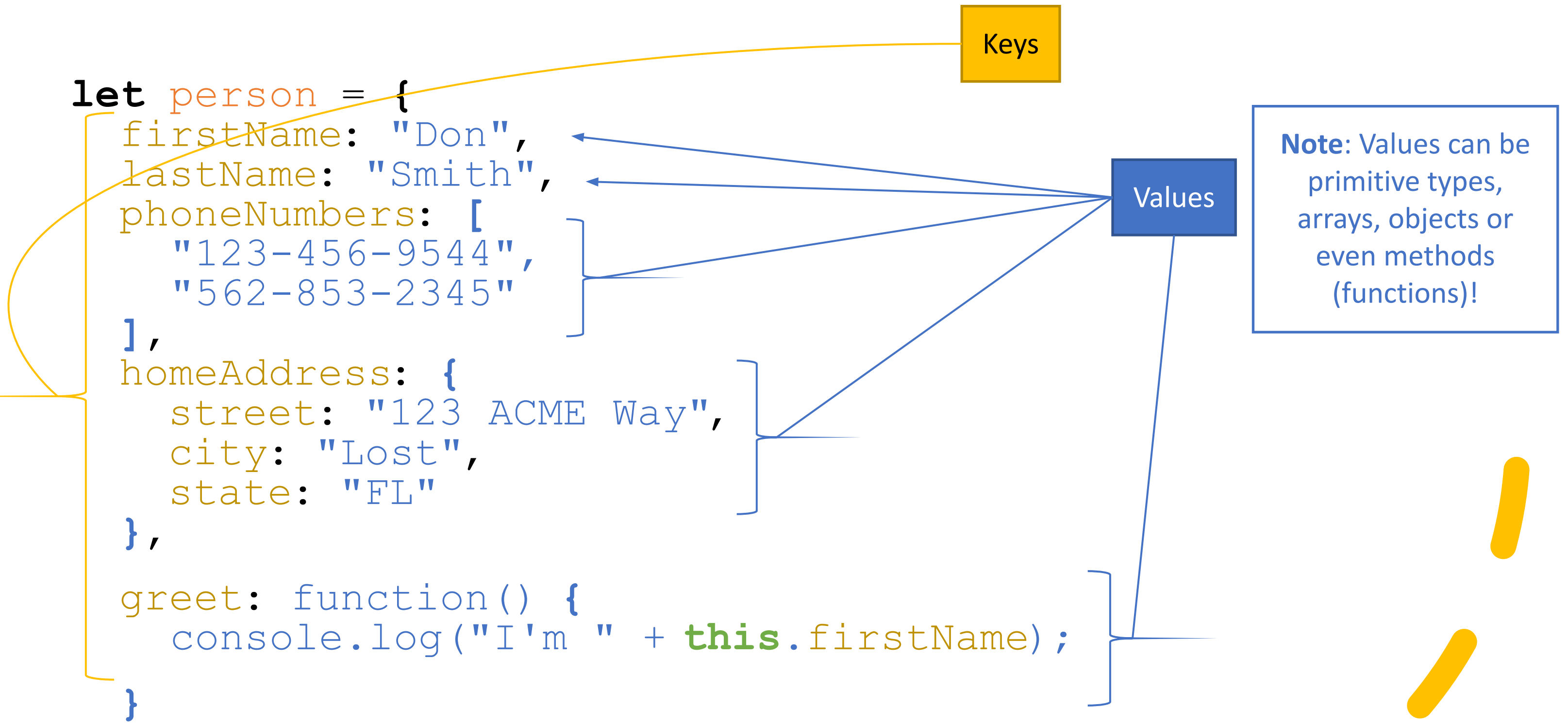


https://**developer.mozilla.org**/en-US/docs/Web/JavaScript/Reference/Global_Objects/**Array**

Objects

- Groups data and functionality together
- Use of key/value pairs



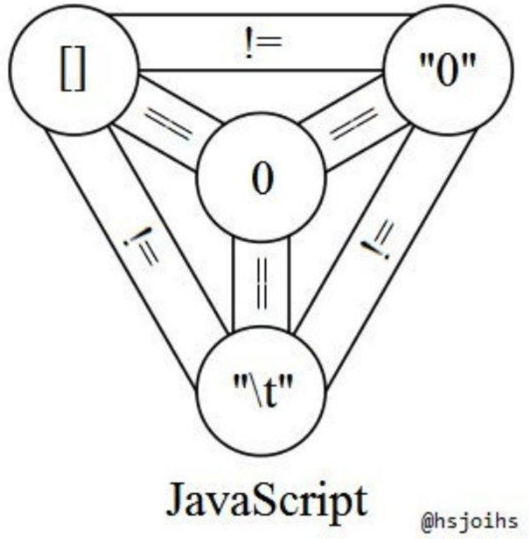


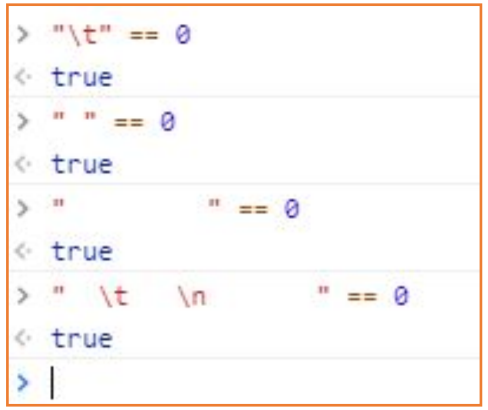
```
let person = {
  firstName: "Don",
  lastName: "Smith",
  phoneNumbers: [
    "123-456-9544",
    "562-853-2345"
  ],
  homeAddress: {
    street: "123 ACME Way",
    city: "Lost",
    state: "FL"
  },

  greet: function() {
    console.log("I'm " + this.firstName);
  }
};
person.greet(); // I'm Don
```

Note: Values can be primitive types, arrays, objects or even methods (functions)!

Equality
(== vs ===)

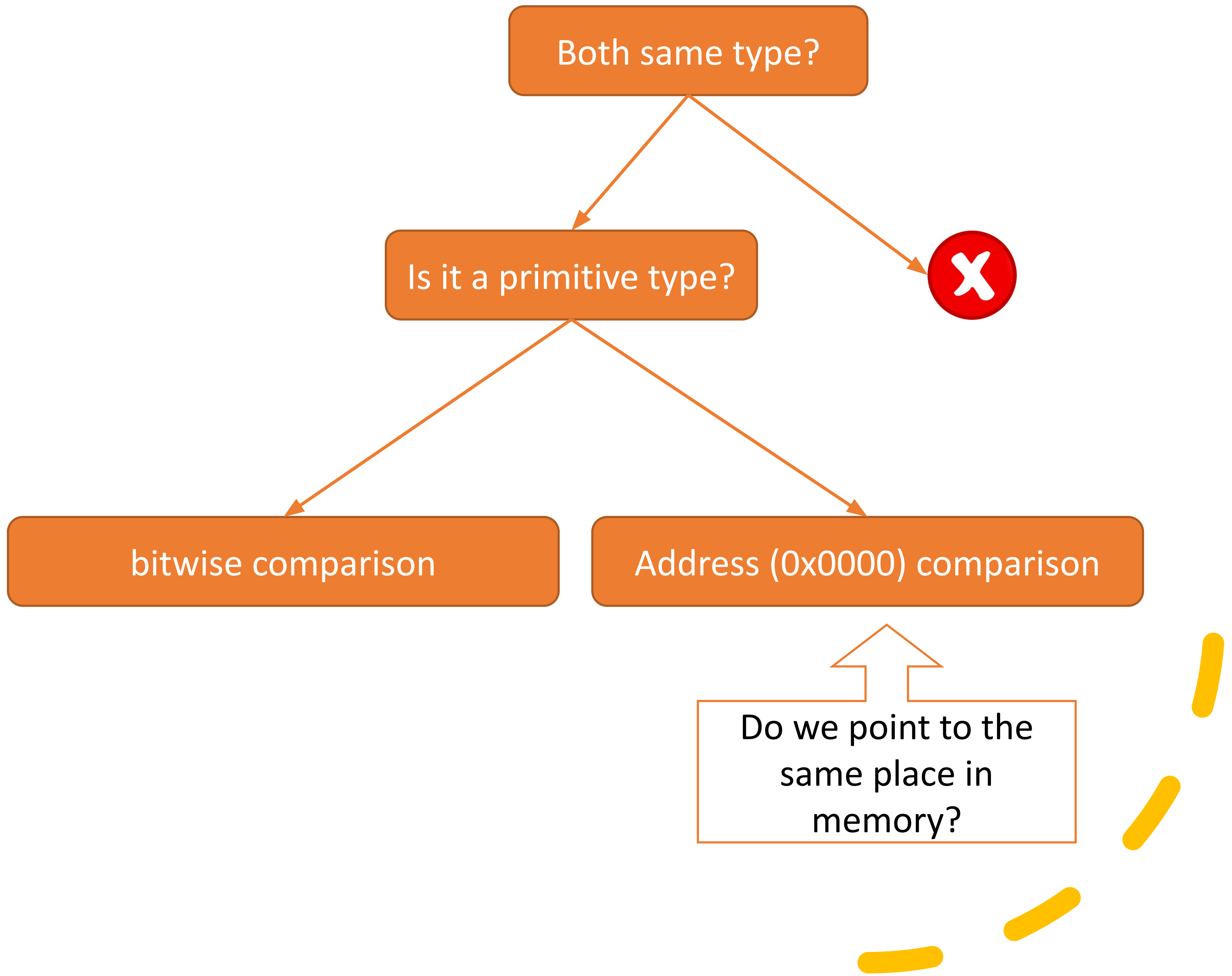
- == vs ===, != vs !==
  - == uses type coercion to convert to a common type, then checks for equality
  - === compares types, then checks for equality





```
> "\t" == 0
<· true
> " " == 0
<· true
> "          " == 0
<· true
> " \t   \n        " == 0
<· true
> |
```

Equality (===)



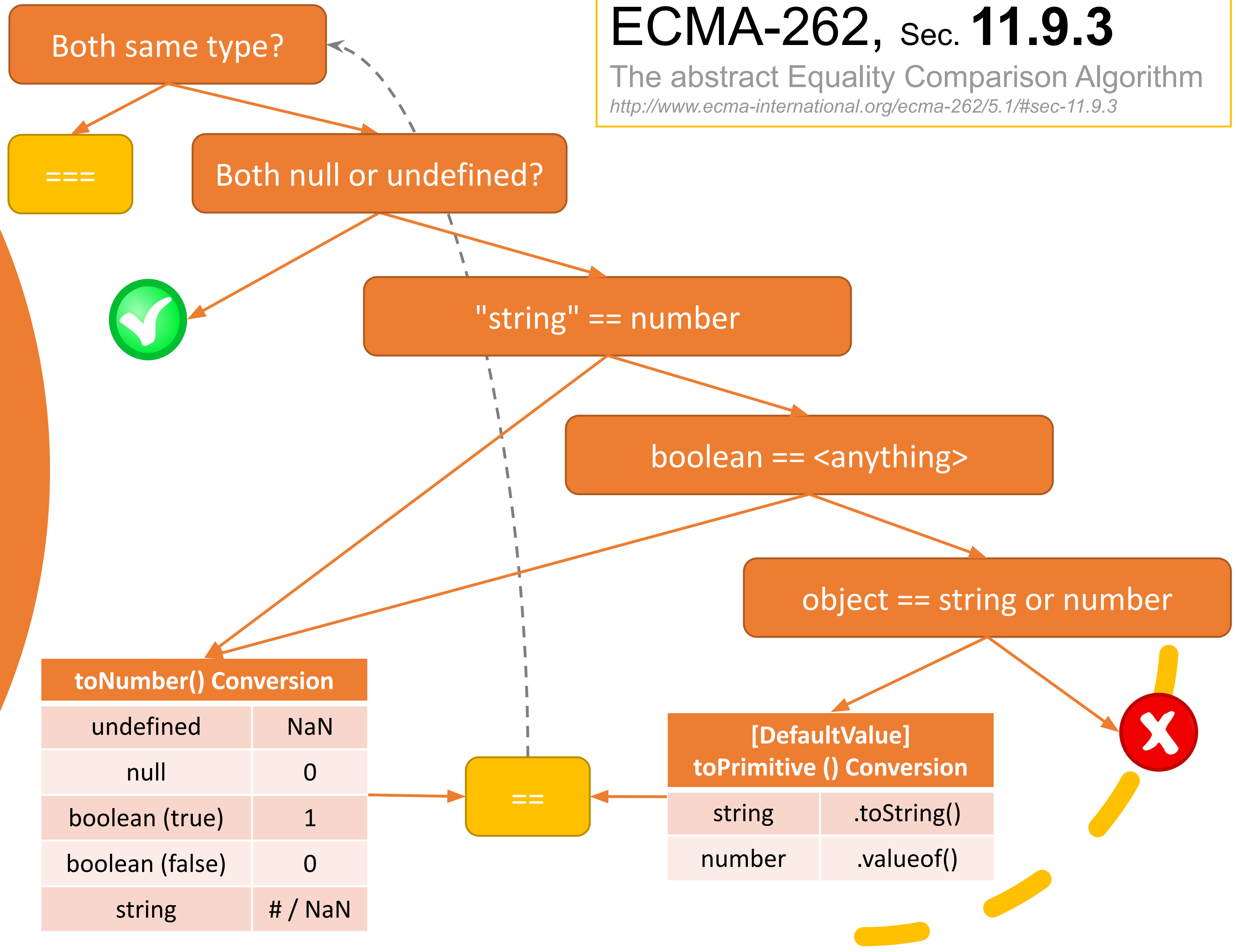
Both same type?
Is it a primitive type?
bitwise comparison
Address (0x0000) comparison
Do we point to the same place in memory?

Equality
(==)

ECMA-262, Sec. **11.9.3**

The abstract Equality Comparison Algorithm

http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3



toNumber() Conversion	
undefined	NaN
null	0
boolean (true)	1
boolean (false)	0
string	# / NaN

[DefaultValue] toPrimitive () Conversion	
string	.toString()
number	.valueof()

EQUALITY

NEVER* use == for equality.
ALWAYS* use === for equality