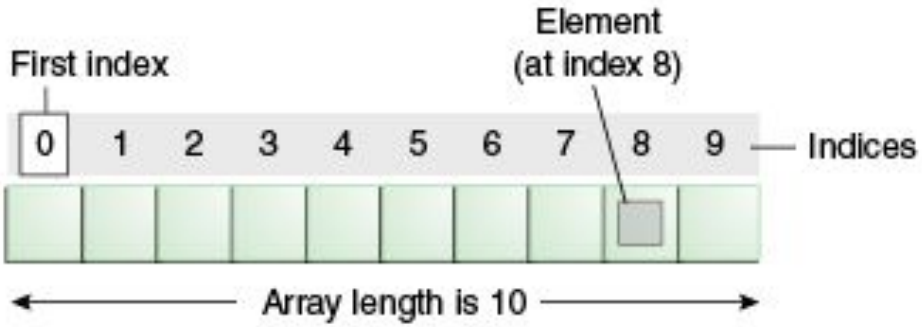
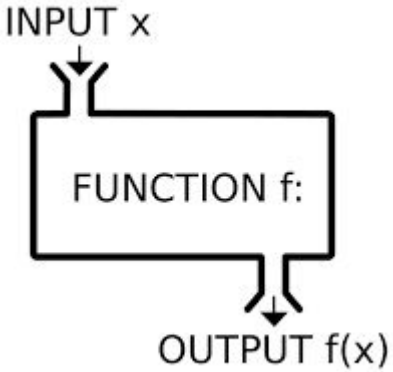
Object , Arrays and Functions

Arrays



Functions



Agenda

Arrays (also objects)

- Collection
- Array index
- Push/unshift
- Pop/shift

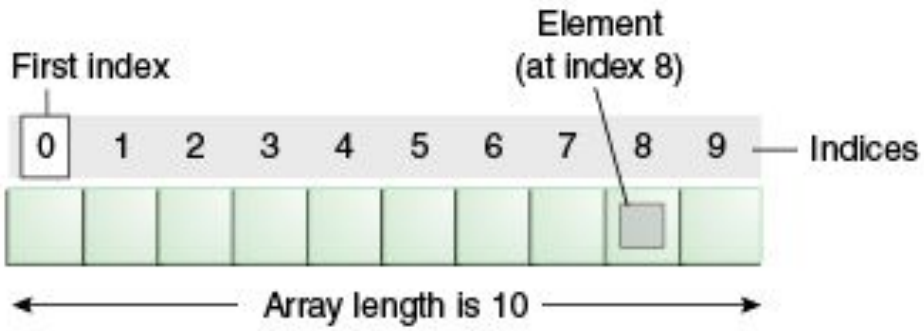
Functions (also objects)

- Anonymous functions
- Invoking a function
- Parameters vs arguments
- Parameter validation + NaN
- Throwing errors

Es6

- Arrow functions
- Array and string functions
- Type checking

Arrays



A data structure consisting of a collection of elements (values or variables), each identified by at least one **array** index.

JS Arrays

- Group same type items.
  They are mostly used to group together different items of the same type.

Track height of a child:

```
var height2000 = 1.20;
var height2001 = 1.24;
var height2002 = 1.31;
var height2003 = 1.40;
```

```
var heights = [1.20, 1.24, 1.31, 1.40];
```

Specify color for a shirt:

```
var color1 = "red";
var color2 = "blue";
var color3 = "yellow";
```

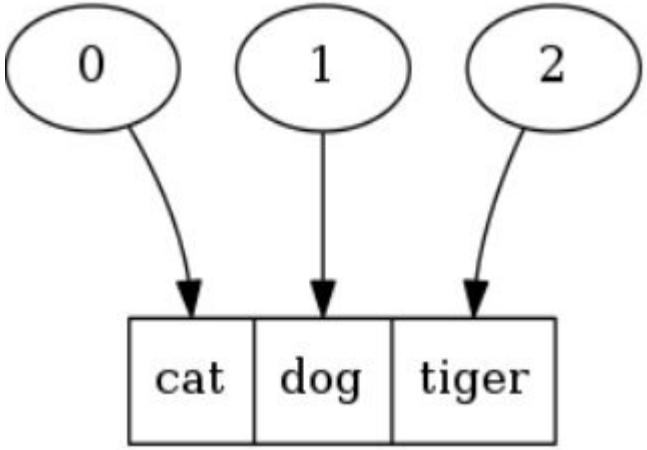
```
var colors = ["red", "blue", "yellow"];
```

- Meaning: items that have the same structure but different values.

JS Arrays

- Ordered collection
  It is a single object that represents a collection of *n* items.

- Just like a string!



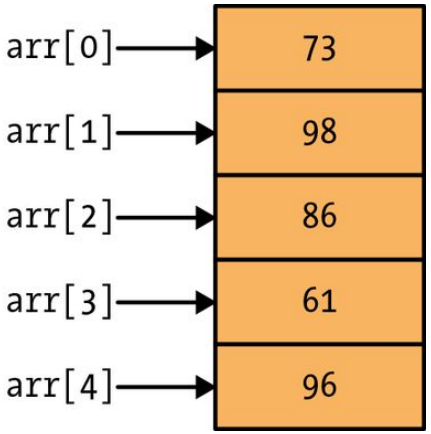
JS Arrays

- Access by key (index)
  Just like a string you can access the array items by key

```
var arr = [4,5,6];
arr[0]; //?
//4

arr[2]; //?
//6
```

[0]	[1]	[2]	[3]	[4]
73	98	86	61	96



Arrays of everything

- An array of all answers (true/false) in a test
  - [true,true,false,true]
- An array of **strings** that has a list of user names
  - [‘bill’,’donna’,’ray’]

Arrays of everything

- Arrays are not sorted, they are ordered.
  - [1,7,4,4]
  - Meaning each number is in a specific place.
  - There is an order (1 before 7) but the order is not the result of some logical sorting (ex. high numbers first)

Type of Arrays

- Arrays can contain any other type, including other arrays. (we will see it later)
- Most of the time you will not want a single array to contain more than one type.

Creating Arrays

- The literal notation for an array is []

```
var emptyArray = [];

var numbers = [44, 5, 63];
```

Referencing items

- We can use an index to reference items in the array

```
var arr = [1,2,3];
var first = ?;
var first = arr[0];//1
var second = ?;
var second = arr[1];//2
```

Array Jokes


Does your dog bite?
No, but it can hurt you in other ways.
Array indexing starts at 1

Referencing items

- Auto growing
  JavaScript arrays are auto growing, meaning that if we try To add an item the an index that does not exist, the array will grow to accommodate.

```
var arr = [1,2];
arr[10] = 100; // no exception thrown
```

- Mutable
  Unlike strings, arrays are mutable, meaning we can add or change items without creating a new array.

Array Length

```
var empty = [];
var numbers = [2,4,6];

What happens if we try to access an element that
doesn't exist?
empty[1];
//undefined

We can get the length of the array:
empty.length;
//0
numbers.length;
//3
```

Push/unshift

- If we want to add one item to an array we can use push or unshift
- The difference is where the new item gets added

```
var arr = ["old", "old", "old"];

arr.push("pushed");
arr; // ["old", "old", "old", "pushed"]

arr.unshift("unshift");
arr; // ["unshift", "old", "old", "old", "pushed"]
```

Pop/Shift

- If we want to remove one item from an array we can use pop or shift

- Returns the removed element
  These two functions also return the item that was removed.

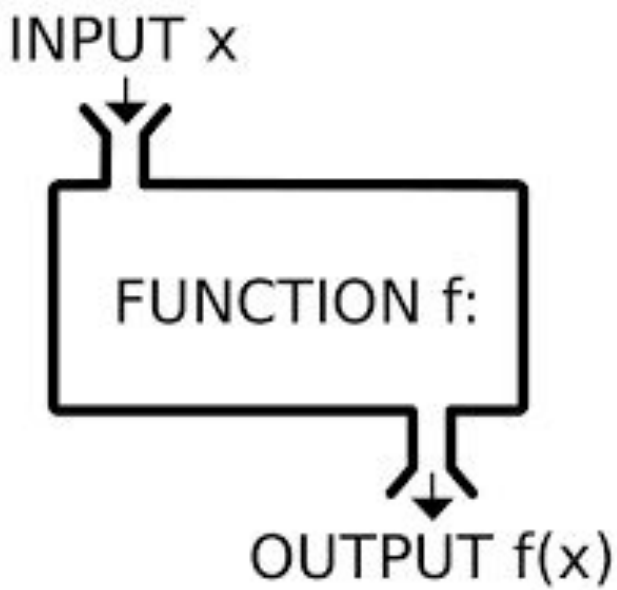
```
var arr = [1,2,3];
var first = arr.shift();//1
var last = arr.pop();//3
```

Questions?

```
console.log("Questions?");
```

Functions

Function Definition:
A function is a set of operations/commands that we execute with given parameters.



Functions can be invoked

In order to use a function, we first need to declare it! Here we declare a super simple function:

```
function print(){
   console.log("I was invoked");
}
```

- Now we can invoke (call) it.
  - When you call a function, the code inside the function body will be executed
- **Invoke**: We invoke a function by putting a parenthesis after the name of the function.

```
print();
```

Functions have a return value

- when the function is done running a value will be returned.

Here we declare another function:

```
function calc3Power2(){
    return Math.pow(3,2);
}
```

In order to return a value we need to use the return saved word.

- When we invoke the function it will return a value.

```
var result = calc3Power2();
// result is now 9
```

Functions have a return value

- After we return a value, the function will stop running.

```
function calc3Power2(){
   console.log("I was invoked");
   return Math.pow(3,2);
   console.log("finished running");
}
```

- What will be printed after we invoke the function?

```
var result = calc3Power2();
```

Functions take parameters

- We can pass parameters to a function so that the function can use them.

Here we declare a function with 2 parameters:

```
function add(a, b){
   return a+b;
}
```

- When we invoke the function we will pass values inside the parenthesis.
- These values are called arguments.
- The arguments are separated by a comma

```
var result = add(2,5); //?
var result2 = add(result, 3); //?
var result3 = add(result, result2); //?
```

Functions take parameters

Note that the names of the arguments is not related to the names of the parameters:

a,b are add's parameters

```
function add(a, b){
   return a+b;
}


var num1 = 2;
var num2 = 3;

var result = add(num1, num2);
```

num1 ,num2 are arguments we pass to add

Parameters vs Arguments

Parameters refers to the list of variables in a method declaration:

```
function add(parameter1, parameter2){
    return parameter1 + parameter2;
}
```

Arguments are the actual values that are passed to a method when it is invoked:

```
var argument1 = 3;
var argument2 = 2;
add(argument1, argument2);
```

Parameters

What is the difference between

```
function add(a, b){
   return a+b;
}

function add2(x, y){
   return x+y;
}
```

What will be the result of:

```
var num1 = 2;
var num2 = 3;
add(num1, num2);

add2(num1, num2);
```

Parameters

- We can specify any number of parameters:

```
function get0(){
   return 0;
}

function add10(number){
   return number + 10;
}

function add(a, b){
   return a+b;
}

function add3(a, b, c){
   return a+b+c;
}
```

Functions are closure

- **Closure**: Variables declared inside a function can only be accessed from within that function.

```
var globalVar = "global";

function foo(){
   console.log(globalVar); //can be accessed inside the function
   var innerVar = "inside a function closure";
}

foo(); // invoking the function

globalVar; //"global"
innerVar; //ReferenceError: innerVar is not defined
```

Creating functions

- Option #1
  - We are creating a function with a name, but with the return value not assigned to a variable
  - The function can be referenced using the name.

```
function foo() {

}
```

Creating functions

- Option #2 – function expressions
  - We are creating a function, not giving it a name, but assigning it to a variable.
  - Functions created this way are called **anonymous functions**.

```
var foo = function(){
    // function logic
}
```

Invoking a function

- We have already called functions in JavaScript.
- Where?
  - the console object has a function in it, called log, that we have been using a lot.

```
console.log('hello world');
```

Functions always return some value

- This function returns a number

```
function a() {
    return 10;
}
```

- This function returns a string

```
function b() {
    return "hello";
}
```

- What will this function return?
  - We did not define a return value!

```
function c() {
    var x  = 5 + 7;
}
```

- When the return value of a function is not defined, the return value will be undefined.

Functions take parameter

- This function gets two parameters, it assumes that they are numbers.
  - If the numbers are both smaller than 10, the numbers are multiplied and the value is returned
  - if they at least one of them is larger, they are added and the value is returned

```
function addOrMultipleNumbers(num1, num2) {
    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

Parameter Validation

- What would happen if we invoked this function but did not pass in two parameters?

- Instead we passed only the first number?

```
addOrMultiply(6);
```

```
function addOrMultipleNumbers(num1, num2) {
    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

Parameter Validation

- If we don't define the value of a parameter when calling a function, the value if the parameter will be…

…Say it with me

!!!Undefined

```
function addOrMultipleNumbers(num1, num2) {
    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

Parameter Validation

- So if num2 is undefined. What will we get back from the function?
  - First we will check if num1 is less than 10, maybe it is.
  - Even if it is, we know for sure that we cannot say that undefined is smaller than 10.

```
function addOrMultipleNumbers(num1, num2) {
    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

Parameter Validation

- So the else clause is going to run.
  - What is 5 + undefined?
  - I can tell you what it is not, **it's not a number.**
  - So we get back a special value 'NaN', lettings us know something unexpected happened.

```
function addOrMultipleNumbers(num1, num2) {
    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

What is NaN

NaN = Not a Number

It's a special value letting us know that we tried to do something mathematical with a value that is not a number.

```
var x = 5 - "hello"; // now x is NaN
```

After we get a NaN in our calculation, it is infectious. Every value that comes into contact with it will also become NaN.

```
var t = x + 10; // now t is also NaN
```

Nan

?How can we check if a variable is NaN

```
?typeof num !== "number"; //what will be the result

:The intuitive way is actually the wrong way
returns false. NaN is actually a type of number//
“typeof NaN !== "number

:So How about
num === NaN
returns false//

:The correct way
num !== num
returns true. finally it works. NaN is the only value that is//
//not equal to itself
NaN !== NaN
```

Read more here

Parameter Validation

- **Throw error**: If we want to be very careful, we can check for an un expected input at the beginning of a function, and throw an error if the input is bad.

- This will make sure the developer calling our function (might even be us) will know they did something wrong

Syntax: `throw new Error("error message");`

```
function addOrMultipleNumbers(num1, num2) {

    if(typeof num1 !== 'number' || typeof num2 !== 'number' ){
        throw new Error('addOrMultipleNumbers expects two numbers ' +
        'but instead got (' + num1  + ', ' + num2+ ')');
    }

    if (num1 < 10 && num2 < 10) {
        return num1 * num2;
    } else {
        return num1 + num2;
    }
}
```

Functions validation

- When should we check the parameters?
- We're not advocating this approach inside every function. It depends on:
  - The cost of getting it wrong, if the cost is high, extra protection is wise.
  - Sometimes you will be required to write more defensive code by your superiors to keep the same code standard.

We need to set boundaries what is more or less crucial and should be validated.

Immediately invoked functions

We already saw something like that:

```
var greeting = function(name){
     return "Hello " + name;
}

console.log(greeting); // ?
// function
console.log(greeting("David")); //?
// Hello David
```

But what is happening here?

```
var greeting = function(name){
     return "Hello " + name;
}("David");

console.log(greeting); // ?
// Hello David
console.log(greeting("David")); //?
//TypeError: greeting is not a function
```

Self invoked functions

A closer look

```
var greeting = function(name){
    return "Hello " + name;
}("David");
```

This is the same function definition we had before.
But what comes after it?

```
var greeting = function(name){
    return "Hello " + name;
} ("David");
```

Function definition

Invoking the function and passing an argument

We invoke the function => the function code is being executed => it returns the string "Hello David" that is stored in the greeting var.

EcmaScript 6

JS History

A Brief History of EcmaScript

1995: JavaScript is born as LiveScript
1997: ECMAScript standard is established
1999: ES3 comes out and IE5 is all the rage
2000–2005: XMLHttpRequest, a.k.a. AJAX, gains popularity in app such as Outlook Web Access (2000), Gmail (2004) and Google Maps (2005).
2009**: ES5 comes out** (this is what most of us use now)
with forEach, Object.keys, Object.create, and standard JSON
2015: **ES6/ECMAScript2015 comes out**; it has mostly syntactic sugar

ES7?

ECMAscript 6 - Motivation

Solves ECMAScript 5 problems

Improved syntax

New features

Browser compatibility

Pay Attention - Browser Compatibility

* Many es6 features are already implemented by the major browsers, but we should always verify the features we use.

* Polyfills – Fallback code. Code that implements an unsupported feature.

* Babel – Transforms es6 to es5

Arrow Functions

Syntax - 1 Parameter

```
// es5
function add1(num) {
  return num + 1;
}
```

```
// es6
x => x + 1;
(x) => x + 1;
(x) => {return x + 1};
```

Arrow Functions

Syntax – Multiple Parameters

```
// es5
function sum(x, y) {
  return x + y;
}
```

```
// es6
(x, y) => x + y; // This returns a value
(x, y) => {return x + y}; // The same.
```

Arrow Functions

Syntax – No Parameters

```
// es5
function return2() {
  return 2;
}
```

```
// es6
() => {return 2};
```

Arrow Functions

Practice:

Convert the following functions to arrow functions:

```
function foo1(x) {
  return x*x;
}

function foo2(x) {
  return x > 5 ? 3 : x;
}

function foo3(x, y) {
  return Math.min(x, y, -1);
}
```

Ternary

A short for our familiar if – else.

Sytax

```
var result = condition ? exp_if_true : exp_if_false;
```

If the condition is true the first value is returned, otherwise the second.

Example:

```
var color = age > 30 ? "gold" : "green";
```

Practice

```
1. age  = 46;
2. age = 12;
```

Arrow Functions

Practice:

Explain what these arrow functions do:

```
v => v % 2 == 0;

(a, b) => (a + b)/2;

v => v * 2;

str => str.charAt(1);
```

Arrow Functions

Practice -Write arrow function that:

- Decrease 10 from a given number
- Calculates the factorial of 5 “(5!)”
- Concats 2 strings

Arrow Functions

Usage

We use arrow functions mostly as anonymous functions
Which means they don't have a name

```
(x) => x * x
```

Most of the time they will be used as arguments for a function

```
function invoke(func) {
   func();
}

invoke(x => console.log(x));
```

Arrow Functions

Usage

```
We can also assign them to a variable:

var sum = (x,y) => x + y;
sum(2,3)
```

Built-in Methods

New Array Methods

Find

```
[1,2,3,4].find(x => x > 2); // 3
```

Find Index

```
[1,2,3,4].findIndex(x => x > 2); // 2
```

Filter

```
[1,2,3,4].filter(x => x > 2); // [3,4]
```

Built-in Methods

New String Methods

Starts With

```
"challenge".startsWith("c"); // true
```

Ends With

```
"challenge".endsWith("e"); // true
```

Includes

```
"challenge".includes("l"); // true
```

Template Literals – String Interpolation

```
var userName = "Bilbo";

// es5
var str = "hello " + userName + " and good
day!";

// es6
var es6_str = `hello ${userName} and good
day!`;
```

Enclosed with a back tick

Can contain placeholder, indicted by dollar sign and curly braces (**${expression}**)

Template Literals – String Interpolation

Another example

```
// es5
var str = "5+6 = " + (5+6).toString();

// es6
var es6_str = `5+6 = ${5+6}`;
```

Function Chaining

Let's think about the following tasks.

For a given array [4,7,3] we want to :

1. Add some new numbers (for example [12,9])
2. Remove the odd numbers.
3. Get the last element in the array

How would our code look like?

```
var arr = [4,7,3];
arr = arr.concat([12,9]);
arr = arr.filter(x => x%2 == 0);
arr.pop();
```

What if I told you I can write it in just 1 line….

Function Chaining

```
var arr = [4,7,3];

arr.concat([12,9]).filter(x => x%2 == 0).pop();
```

[4,7,3,12,9] [4,12] 12

Since `concat` returns an array we can immediately call `filter` which is another array function.

Function Chaining

```
var arr = [4,7,3];
```

1 2 3

```
arr.concat([12,9]).filter(x => x%2 == 0).pop();
```

[4,7,3,12,9] [4,12] 12

These functions will be called one after the other, from left to right.

Function Chaining

Another Example

How about now?
What can I chain to the following expression:

```
"I love ".concat("it!").???
```

What can we chain next?

```
One example:

“I love “.contact(“it!”).toUpperCase()

// Result: “I LOVE IT!”
```

Summary

- You needed to understand:
  - The different ways to access object properties
  - The different ways to define functions
  - How functions handle missing variables
- You need to remember:
  - What is an object
- You need to be able to do:
  - Use arrays, object and functions in JS

Questions?

```
console.log("Questions?");
```

Cheat Sheet

Arrays

Create:
```
var empty = [];
var numbers = [1,6,3];
```
Get value:
```
numbers[0]; // 1
numbers[2]; // 3
```
Length: `numbers.length; //3`

Updating:
```
numbers[5] = 1; //arrays auto grow
numbers.push(6);//[1,6,3,1,6]
numbers.unshift(4)//[4,1,6,3,1,6]
numbers.pop(); //6
numbers.shift();//4
```

Array functions

```
[1,2,3].find(x => x>2);//3
[1,3].findIndex(x => x>2);//1
[1,3,4].filter(x => x>2);
```

String Functions

```
"go".startsWith("g"); // true
"do".endsWith("o"); // true
"foo".includes("oo"); // true
```

Template Literals

```
var es6_str = `5+6 = ${5+6}`;
```

Functions

```
function add(parameter1, parameter2){
    return parameter1 + parameter2;
}
var argument1 = 3;
var argument2 = 2;
add(argument1, argument2);
```

Anonymous:
```
var oneWay = function(){
    // function logic
}
```

Arrow functions

Multiple parameters:
```
(x, y) => x + y;
(x, y) => {x + y};
```
No parameters:
```
() => {2};
```
1 parameter:
```
x => x + 1;
(x) => x + 1;
(x) => {x + 1};
```

Type Checking
```
isNaN(x);
isFinite(4/0);
```

Ternary:

Syntax: `var result = condition ? if_true : if_false;`

Example: `var color = age > 30 ? "gold" : "green";`