ES2015 (ES 6) Exercises

EX-0

Skim this document, most of the examples below are taken from here. Use it as a “cheat sheet” for es2015.

EX1

Execute the examples from (1) related to *Constants* and *Scoping*. Make sure you understand and can explain block-scoped variables in relation to ES5 (var) scoping and hoisting.

In all the following exercises you should use *let-declarations*, unless you really have a reason for not doing so.

EX2

a1) Add the necessary (missing) code to make this example work:

a2) Why does this work?:

```
var odds = evens.map(v => v+1);
```

while this doesn't (fix the example below, without going back to the solution above)?

```
var odds = evens.map(v => {
  v+1
});
```

EX3 arrow functions and this

A)
For this exercise you should refer to this slide (http://slides.mydemos.dk/javascript1/js.html#19) as a reference to ES5 *this*-pitfalls.
Use the Constructor function in the example below, to explain about the ES5 *this*-behaviour. Execute the example, and solve the problem, first using ES5 features, and then using an es2015 arrow function.

```
function Numbers(numbs) {

  this.nums = numbs;
  this.fives = [];
  this.nums.forEach(function (v) {
    if (v % 5 === 0) {
      this.fives.push(v);
    }
  });
}
var numbers = new Numbers([1,3,5,10,14,20,33,50]);
console.log(numbers.fives);
```

B) Arrow functions and *this* or when not to use arrow functions

This example (taken from the slide referred to above), shows how we "loose" this, when extracting a method from an object.

```
var counter = {
    count: 0,
    inc: function () {
      this.count++;
    }
  }
var func = counter.inc; //Store "reference" to inc
func();
console.log(counter.count); //Still zero
counter.inc();
console.log(counter.count); //Now one
```

Rewrite the `inc()` function to use the arrow notation, and test whether this; solves the problem, makes it worse or leaves it unchanged.

Ref: Do ES6 Arrow Functions Really Solve “this” In JavaScript?

EX-4 Template literals

Execute this example And use template literals whenever it makes sense for all the following exercises.

EX-5 - Rest Parameter and the spread operator

A) Implement the function `f(..)` below:

```
        function f(x,y,...rest){
          ...
        }
```

So this statement:

```
        console.log(f(5,2,true,2,"hello World",[1,2,3],new Date(),{}));
```

Will produce this output (should obviously work for any number/type of arguments):

```
Sum: 7
rest value 1 is a: Boolean
rest value 2 is a: Number
rest value 3 is a: String
rest value 4 is a: Array
rest value 5 is a: Date
rest value 6 is a: Object
```

Hint: With es2015 you can get the class name using this construct: `myinstance.constructor.name`

B) Test the rest operator using the code below:

```
var rest = [true,2,"hello World",[1,2,3],new Date(),{}];
var restParams = [ ...rest];
console.log(f(5,2,...restParams));
```

C) What will this line produce? `var chars = [... f(5,2,...restParams)];`

EX-6

Assuming we had these variables (for example passed in via a HTTP request):

```
let fName  = "Kurt";
let lName  = "Wonnegut";
let age = 98
```

Create an object, using the *Property Shorthand notation* with a `fName, lName` and `age` property.

EX7 Destructing Assignment

A) Given these declarations: `let fName = "Kurt", lName = "Wonnegut";`

Implement a one-liner (using Array matching) to swap the two values so this statement:
`console.log(`First: ${fName}, Last: ${lName}`);`

Will print: `First: Wonnegut, Last: Kurt`

B) Given the method below

```
function getPerson(){
  return {
    firstName: "Kurt",
    lastName: "Wonnegut",
    gender : "Male",
    email: "kurt@wonnegut.dk",
    phone: "12345",
  }
}
```

Implement a one-liner (using the object matching shorthand notation) that will initialize (only) two variables `lastName` and `phone`.

EX-8 – ES2015 Modules

Rewrite the *f(..)* method from EX5 into a reusable es2015 module, and import the function into a new file and test.

Hint: *Since you are using Node, Node will think of this as one of its own modules and look into node_modules for the module unless you do the usual "./myModule" for your own modules*

EX9 Classes and Inheritance with es2015

A) The declaration below defines a Shape class, whicha as it's only properties has a `color` field + a `getArea()` and a `getPerimeter()` function which both returns undefined. This is the closest we get to an abstract method in Java.

```
class Shape {
  constructor(color){
    this._color = color;
  }
  getArea() {
    return undefined;
  }
  getPerimeter() {
    return undefined;
  }
}
```

Provide the class with a nice (using template literals) `toString()` method + a getter/setter for the colour property. Test the class constructor, the getter/setter and the two methods.

B) Create a new class Circle that should extend the Shape class.

Provide the class with:

- A radius field
- A constructor that takes both colour and radius.
- Overwritten versions of the three methods defined in the Base
- Getter/Setter for radius

Test the class constructor, the getters/setters and the three methods.

C) Create a new class Cylinder (agreed, not a perfect inheritance example) that should extend the Circle class.

Provide the class with:

- A height field
- A constructor that takes colour, radius and height.
- Overwritten versions of the three methods defined in the Base (`getPerimeter()` should return undefined)
- A `getVolume()` method
- Getter/Setter for height

Test the new class

D) The getX() methods (getArea(), getPerimeter() and getVolume()) are all candidates for a getter.

Rewrite the three methods to use the getter syntax; that is `console.log(circle.radius)` instead of `console.log(circle.getRadius())`

These exercises are postponed to period-5

~~EX10 - the iteration Protocols~~

~~Skim the sections related to Iteration Protocols and implement (and understand) the two examples from the slides:~~

~~http://js-plaul.rhcloud.com/es2015_typescript/es5VStypescript.html#8~~

~~EX11 Generators~~

~~A) Skim these sections related to Generators and implement a few of the simple examples:~~

- ~~Iterators and generators~~
- ~~Generator~~

~~B) Complete the Generator function below (it's not yet a generator function, what is missing)~~

```
~~function* makeNames() {~~

~~  let firstNames = ["Lars", "Jan", "Ida", "Tine","Thomas"];~~
~~  let lastNames = ["Mortensen","Peterson","Obama","Jensen","Hansen"];~~

~~}~~
```

~~So these statements~~

```
~~let index = 0;~~
~~for(let name of makeNames()){~~
~~  console.log(name);~~
~~  if(index++ === 50){~~
~~     break;~~
~~  }~~
~~}~~
```

~~Will produce 50 objects as sketched below:~~

```
~~{ firstName: 'Ida', lastName: 'Jensen' }~~
~~{ firstName: 'Tine', lastName: 'Jensen' }~~
~~{ firstName: 'Jan', lastName: 'Mortensen' }~~
~~{ firstName: 'Lars', lastName: 'Jensen' }~~
~~{ firstName: 'Lars', lastName: 'Peterson' }~~
~~{ firstName: 'Tine', lastName: 'Peterson' }~~
~~{ firstName: 'Ida', lastName: 'Peterson' }~~
~~{ firstName: 'Tine', lastName: 'Peterson' }~~
~~{ firstName: 'Jan', lastName: 'Obama' }~~
~~{ firstName: 'Ida', lastName: 'Obama' }~~
…
```