ES6/ES2015 and Beyonds with the Best Practices

Short History

- Want to added support for Java applets more accessible to non-Java programmers in Netscape.
- Developed by Brendan Eich of Netscape
- December 1995
- Mocha => LiveScript => JavaScript
- Popular!
- Microsoft release Jscript
- NetScape submit to ECMA for standardize

ES6 Basic

- /* Comment */
- // Comment
- console.log('print out');
- Syntax mostly derived from C language
  - if (true){}
  - while (true){}
  - do{…} while(true);
  - for(init ; condition; incr) {}

Primitive Types

- Directly Store Value
  - String
  - Number
  - boolean
  - Null
  - undefined

Primitive Type #1: String, Number and Boolean

```
1
2   const myString = 'myString'; //Use single quote
3   console.log(myString + 'is a myString'); // => myString is a myString
4   console.log(typeof(myString)); // => string
5
6   const number1 = 3;
7   const number2 = 3.33;
8   console.log(number2 - number1); // => 0.33000000000007
9   console.log(typeof(number1)); // => number
10  console.log(typeof(number2)); // => number
11
12  const boolean = true;
13  console.log(!boolean); // => false
14  console.log(typeof(boolean)); // => boolean
15
```

Primitive Type #2: Null, Undefined, and Reference by Value

```
16  const empty = null;
17  let unknown;
18  console.log(unknown); // => undefined
19  console.log(empty == unknown); // => true
20  console.log(!!(empty)); // => false
21  console.log(!!(unknown)); // => false
22  console.log(typeof(empty)); // => object
23  console.log(typeof(unknown)); // => undefined
24
25  let number3 = number2; // work on its directly value
26  number3 = 10;
27  console.log(number2, number3); // => 0.33 10
```

Exercise 1

```
1
2   const number1 = 5;
3   const number2 = 10;
4   console.log(number2 - number1);
5   console.log(typeof(number1));
6
7   console.log(null == undefined);
8   console.log(!!true);
9   console.log(!!!true);
10  console.log(!!!1);
11  console.log(!!undefined);
12
```

Complex Types

- Reference of its value
  - Object
  - Array
  - Function

Complex Type #1: Object & Array

```
1
2   // Object
3   const myObject = {a: 1, b: 2};
4   console.log(myObject['a']); // => 1
5   console.log(myObject); // => {a: 1, b: 2}
6   console.log(typeof(myObject)); // => object
7
8   // Array
9   const myArray = [1, 2];
10  console.log(myArray[0]); // => 1
11  console.log(myArray); // => [1, 2]
12  console.log(typeof(myArray)); // => object
13
```

Complex Type #2: Function and Pass by Reference

```
1
2  // Function
3  const myFunction = function() {
4      console.log('myFunction');
5  }
6
7  console.log(myFunction); // => [Function: myFunction]
8  console.log(typeof(myFunction)); // => function
9
10 // Pass by Reference
11 const myArray2 = myArray;
12 myArray2[0] = 9;
13 console.log(myArray[0], myArray2[0]); // => 9, 9
14
```

Exercise 2

```
1
2   const myObject = {a: 1, b: 2};
3   console.log(myObject['a']);
4   console.log(myObject['b']);
5   console.log(myObject);
6
7   const myObject2 = myObject;
8   myObject['a'] = 999;
9   console.log(myObject['a'], myObject2['a']);
10
```

Declaration (var, let and const)

- **var, let, const** = Making Variable Declaration
- **const** = Constant Declaration, Can't Reassign
  - Use with reference that never change.
  - Block-level Scope
  - Safer (If reassignment happen, it will throw the errors)

Declaration (var, let and const)

- **let** = Variable Declaration, Can Reassign
  - Block-level Scope
- **var** = Variable Declaration, Can Reassign
  - Global-level Scope
  - Old JavaScript
  - **Dangerous. Don’t use it**

Declaration #1: Global Scope, Block Scope and Constant

```
1
2  const PI = 3.14;
3  PI = 3.15; // => Throw: Assigment to constant variable error!
4  const person = {};
5  person['a'] = 1; // OK!, Adding the key to object, don't change
6                   // the reference of person variable.
7
8  if (true) {
9      const a = 2;
10     let b = 3;
11     var c = 4;
12     console.log(a); // => 2
13     console.log(b); // => 3
14     console.log(c); // => 4
15 }
16
17 console.log(a); // => Throw: a is not defined
18 console.log(b); // => Throw: b is not defined
19 console.log(c); // => 4
```

Declaration #2: General Use Case in For loop

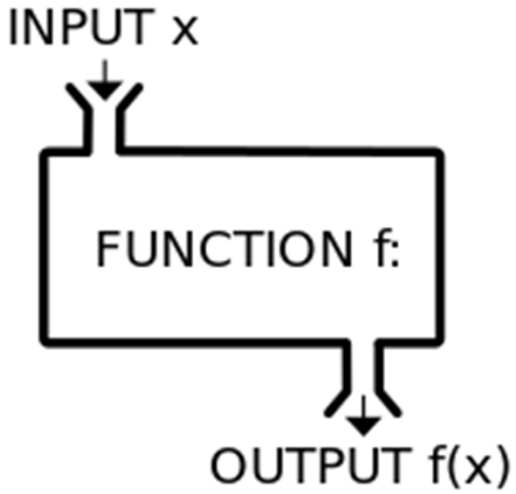
```
21  for (var i = 0; i < 3; i++) {
22      console.log(i); // => 0 1 2
23  }
24
25  console.log(i); // => 2 (Most of people don't want this behavior)
26
27  for (let l = 0; l < 3; l++) {
28      console.log(l); // => 0 1 2
29  }
30
31  console.log(i); // => Throw: l is not defined
```

Exercise 3

```
1
2   const b = 777;
3   b = 888;
4
5   var a = 3;
6   for (let a = 0; a < 2; a++) {
7       console.log(a);
8   }
9   {
10      let a = 999;
11      console.log(a);
12  }
13  {
14      console.log(a);
15  }
16
```

Functions

- Function in JavaScript are objects, which can use as arguments.
- Function can be invoked



Function #1: Declaration, Invocation

```
1
2  // Declaration
3  const add = function(a,b) {
4      return a + b;
5  }
6
7  function add2(a,b,c) {
8      return a + b + c;
9  }
10
11 function printSomething() {
12     console.log('print');
13 }
14
15 // Invocation
16
17 printSomething(); // => print
18 console.log(add(1,2)); // => 3
19 let sum = add2(1,2,3);
20 console.log(sum); // => 6
21
```

Function #2: ES6 Feature - Declaration

```
21
22  // ES6: Declaration, Arrows are function shorthand {=>}
23  const add3 = (a,b) => {
24      console.log('a + b = ', a + b);
25      return a + b;
26  }
27  // If there is only a return statement, you can remove {}
28  // and return keyword
29  const add1 = (a) => a + 1;
30
31  console.log(add3(1,2)); // => 3
32  console.log(add1(1)); // => 2
33  console.log([1,2,3].map(add1)); // => [2, 3, 4]
34
```

Exercise 4

```
1
2   const fn1 = (a) => a * 3;
3   function fn2(a) {
4       return a / 5;
5   }
6   let fn3 = function(a,b) {
7       return a + b;
8   }
9
10  console.log(fn3(fn1(fn1(2)), fn1(fn2(10))));
11
```

Function #3: Anonymous Function

```
1
2  // Anonymous Function: A function has no name (GoT Fan?)
3  setTimeout(
4      function() {
5          console.log('10 ms passed');
6      }
7  , 10);
8  setTimeout(
9      () => { //
10         console.log('100 ms passed');
11     }
12 , 100);
13
14 // Invorked anonymous function immediatelly.
15 console.log('Hey!'); // => Print 'Hey' immediatelly
16
17 (function() { // Encapsulate with anonymous function.
18     console.log('Hey!'); // => Print 'Hey' immediatelly
19 }());
20
21 (() => { // Encapsulate with anonymous function (short hand).
22     console.log('Hey!'); // => Print 'Hey' immediatelly
23 })();
24
```

Function #4: Recursive

```
1
2  // Recursive
3  const finonacci = (a) => {
4      if (a < 1) return 1;
5      return finonacci(a-2) + finonacci(a-1);
6  }
7
8  console.log(finonacci(3)); // => 5
9
```

Function #5: Callback

```
1
2  // Function can be used as arguments (Callback)
3  let callback1 = (total) => {
4      console.log('Oh! Already finished! Total: ', total);
5  }
6  let callback2 = (total) => {
7      console.log('Wait so long. Boring. Total: ', total);
8  }
9
10 let process = (cb) => {
11     let total = 0;
12     for (let i = 0; i < 100; i++) {
13         total += 1;
14     }
15     cb(total); // Call the callback when the long operation done.
16 }
17
18 process(callback1); // => Oh! Already finish! Total: 4950
19 process(callback2); // => Oh! Already finish! Boring. Total: 4950
20
```

Exercise 5

```
1
2  const recurAdd = (a) => {
3      if (a == 0) return 0;
4      return recurAdd(a-1) + a;
5  }
6
7  setTimeout(() => {
8      console.log(recurAdd(10));
9  }, 1000);
10
11 setTimeout(() => {
12     console.log(recurAdd(5));
13 }, 100);
14
15 setTimeout(() => {
16     console.log(recurAdd(1));
17 }, 1);
18
```

Function #6: Exceptions

```
1
2  // Exception: Mishap operations protector.
3  let strongCheckAdd1 = (a) => {// Callee function
4      if (typeof(a) !== 'number') {
5          throw {
6              name: 'TypeError',
7              message: 'add1 needs numbers'
8          }; // Throw error to the caller function.
9      }
10     return a + 1;
11 }
12
13 var try_it = () => {// Caller function
14     try {// Try the following statements
15         strongCheckAdd1('abc');
16     } catch (e) {// If it throws some error!, CATCH!
17         console.log('Error: ', e); // print it out.
18     }
19 }
20
21 try_it(); // => Error: { name: 'TypeError'
22           // message: 'add1 needs numbers' }
23
```

Function #7: ES6 Feature - Rest Params (Spreads ...), Default Params

```
1
2  // ES6: Rest Params (Spreads ... parameters)
3  function concatenateAll(...args) {
4      console.log(args[0], args[1]); // => Hey I
5      console.log(args); // => 'Hey', 'I', 'kinda', 'miss', 'you'
6      console.log(args.length); // => 5
7      return args.join('');
8  }
9  console.log(concatenateAll('Hey', 'I', 'kinda', 'miss', 'you'));
10 // => HeyIkindamissyou
11
12 // ES6: Default parameters
13 function addWithDefaults(a = 0, b = 0) {
14     return a + b;
15 }
16 console.log(addWithDefaults()); // => 0
17
```

Function #8: ES6 Feature - Named Parameters

```
1
2  // ES6: Named params
3  function rectArea ({ width = 5, height = 5 } = {}) {
4      return width + height;
5  }
6
7  console.log(rectArea()); // 25
8  console.log(rectArea({})); // 25
9  console.log(rectArea({width: 3, height: 3})); // 90
10
```

Function's Airbnb Style Guides

- Style Guide #1. Use function declarations instead of function expressions.

```
// bad
const foo = function() {
};

// good
function foo() {
}
```

Style Guide #2

- Wrap immediately invoked function expressions in parentheses.

```
// immediately-invoked function expression (IIFE)
(function() {
    console.log('Welcome to the Internet. Please follow me.')
}());
```

Style Guide #3

- Never declare a function in a non-function block (if, while, etc.)

```
for (let i = 0; i < l; i++) {
    // Bad
    function hey(text) {
        console.log(text);
    }
    hey(i);
}
```

```
// bad
if (currentUser) {
    function test() {
        console.log('Nope.')
    }
}

// good
let test;
if (currentUser) {
    test = () => {
        console.log('Yup.');
    };
}
```

Style Guide #4. Never use arguments, use … instead.

```
// bad
function concatenateAll() {
    const args = Array.prototype.slice.call(arguments);
    return args.join('');
}

// good
function concatenateAll(...args) {
    return args.join('');
}
```

Style Guide #5

- Use parameter syntax rather than mutating function arguments.

```
function handleThings(opts) {
    // No! We shouldn't mutate function arguments.
    // Double bad: if opts is falsy it'll be set to an object which may
    // be what you want but it can introduce subtle bugs.
    opts = opts || {};
    // ...
}

// still bad
function handleThings(opts) {
    if (opts === void 0) {
        opts = {};
    }
    // ...
}

// good
function handleThings(opts = {}) {
    // ...
}
```

HIENLTH

Style Guide #6. Avoid side effects with default parameters

```
var b = 1;
// bad
function count(a = b++) {
    console.log(a);
}
count();  // 1
count();  // 2
count(3); // 3
count();  // 3
```

Style Guide #7: Always put default parameters last

```
// bad
function handleThings(opts = {}, name) {
    // ...
}

// good
function handleThings(name, opts = {}) {

}
```

Style Guide #8:

- Use spread operator … to call variadic function.

```
// bad
const x = [1, 2, 3, 4, 5];
console.log.apply(console, x);

// good
const x = [1, 2, 3, 4, 5];
console.log(...x);

// bad
new (Function.prototype.bind.apply(Date, [null, 2016, 08, 05]));

// good
new Date(...[2016, 08, 05]);
```

Style Guide #8:

- Use Arrow Functions when passing an anonymous function.

```
// bad
[1, 2, 3].map(function(x) {
    const y = x + 1;
    return x * y;
});

// good
[1, 2, 3].map(function(x) {
    const y = x + 1;
    return x * y;
});
```

Style Guide #9

- Omit braces and use the implicit return, if the function body consists of a single expressions.

```
[1, 2, 3].map(number => {
    const nextNumber = number + 1;
    `A string containing the ${nextNumber}`;
});

// good
[1, 2, 3].map(number => `A string containing the ${number}`)

// good
[1, 2, 3].map(number => {
    const nextNumber = number + 1;
    return `A string containing the ${nextNumber}`;
});

// good
[1, 2, 3].map((number, index) => ({
    index: number
}));
```

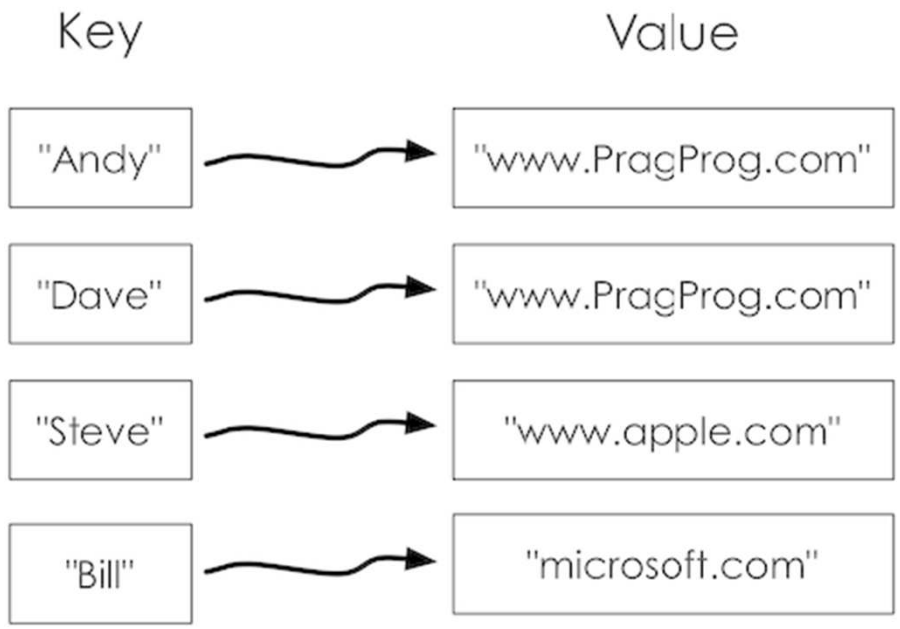
Style Guide #10

- If your function takes a single argument and doesn't use braces, omit the parentheses. Otherwise, always include parentheses around arguments.

```
// bad
[1, 2, 3].map((x) => x * x);

// good
[1, 2, 3].map(x => x * x);

// good
[1, 2, 3].map(number => {
    `A long string with the ${number}`. It's so long that we've broken it
});

// bad
[1, 2, 3].map(x => {
    const y = x + 1;
    return x * y;
});

// good
[1, 2, 3].map((x) => {
    const y = x + 1;
    return x * y;
});
```

Object

- **Object** is a dictionary data structure
- Map between **Key => Value**



Object #1: Declaration, Reference

```
1
2  //Declaration
3  let obj = { key: 2, 'some-thing': 3 };
4  const obj1 = {};
5  const obj2 = {
6      a: 1,
7      b: 'hello',
8      nested: {
9          a: 'a',
10         b: 'b'
11     }
12 };
13
14 // Reference
15 obj = { b: 2 };
16 console.log(obj); // => { b : 2 }
17
```

Object #2: Assignment, Retrieval, Re-Assignment

```
1
2  obj['c'] = 3;
3  console.log(obj); // => { b : 2, c: 3 }
4
5  // Retrieval
6  obj['c'] // => 3
7  obj.c // => 3
8  const sum = obj['c'] + obj['b']; // d = 5
9
10 // Re-Assignment
11 obj['c'] = sum;
12 obj.e = 4;
13 console.log(obj); // => { b: 2, c: 5, e: 4 }
14
```

Object #3: Deletion, Keys, Size, Loops

```
1
2  // Deletion
3  delete obj.c;
4  console.log(obj); // => { b: 2, e: 4}
5
6  // Getting array of keys
7  console.log(Object.keys(obj)); // => [ 'b', 'e' ]
8
9  // Getting size
10 console.log(Object.keys(obj).length); // => 2
11
12 // Loops
13 for (let i in obj) {
14     console.log(i, obj[i]); // => b 2
15                             // => e 4
16 }
17
```

Exercise 7

- Complete the following code that can change digit to reading word.
- For example, 12.3 => “one two dot three”

```
1
2   let number = '123.34'
3   let numToWord = {
4       '1': 'one',
5       '2': 'two'
6   }
7
8   for (var i = 0; i < number.length; i ++) {
9       console.log(number[i]);
10  }
11
```

Exercise

- (1) Write down a function that sum every element in array. E.g. sumArray([12,3,4,1,2,3]) = 25

Exercise

- (2) Write function that count word size case-insensitively.
- Input: "Hello world hello hello earth earth" (Not limited to these word, it can be any words)
- Output: Object{hello : 3, world : 1, earth : 2 }

Object #4: Pass by Reference

```
1
2  let obj = { 'b': 2, 'e': 4 };
3  let obj3 = obj;
4  console.log(obj); // => { 'b': 2, 'e': 4 }
5  console.log(obj3); // => { 'b': 2, 'e': 4 }
6  obj3.e = 5;
7  console.log(obj); // => { 'b': 2, 'e': 5 }
8  console.log(obj3); // => { 'b': 2, 'e': 5 }
9
10 // Functional Pass by Reference: They are never copied.
11 let obj4 = { a: 1 };
12 let num = 1;
13 set0(obj4, num);
14 function set0(obj, num) {
15     obj['a'] = 0;
16     num = 0;
17 }
18 console.log(obj4); // => { a: 0 }
19 console.log(num); // => 1
20
```

Object #5: Clone a Object

```
1
2  // How to clone object? Object can be clone
3  // Object.assign(target, source) => copy members
4  // from source to target and also return target
5  let obj5 = {};
6  Object.assign(obj5, obj);
7  let obj6 = Object.assign({}, obj);
8  let obj7 = Object.assign({ x: 123 }, obj);
9  obj5.z = 'Test';
10 obj6.y = 'Hey';
11 console.log(obj); // => { b: 2, e: 5 }
12 console.log(obj5); // => { b: 2, e: 5, z: 'Test' }
13 console.log(obj6); // => { b: 2, e: 5, y: 'Hey' }
14 console.log(obj7); // => { x: 123, b: 2, e: 5 }
15
```

Object #6: Functions in Object

```
1
2  // Object can hold function, can refer to the
3  // object's members by 'this'
4  let obj8 = {
5      a: 5,
6      b: 6,
7      c: 7,
8      addWithA: function(addition) {
9          return this.a + addition;
10     },
11     addWithAAndSomeVar: function(addition, someVar) {
12         return this[someVar] + this.addWithA(addition);
13     }
14 }
15 console.log(obj8.addWithA(5)); // => 10
16 console.log(obj8.addWithAAndSomeVar(5, 'b')); // => 16
17 console.log(obj8.addWithAAndSomeVar(5, 'c')); // => 17
18
```

Object #7: ES6 Features - Dynamic Keys & Function Shorthand

```
1
2  // ES6: Dynamic Key & Function Shorthand
3  function addValue(value) {
4      return value + 1;
5  }
6  let test = {
7      ['a'+1+'b']: 'a',
8      [addValue(3)]: 'b',
9      // addValue(3):'c', // => Throw: Unexpected number errors,
10                          // all dynamic key must be inside []
11     fn: function() {
12         return 'test'
13     },
14     fnShortHand() {
15         return this.fn() + ' short hand';
16     }
17 };
18 console.log(test); // => { '4': 'b', a1n: 'a', fn: [Function],
19                    //       fnShortHand: [Function: fnShortHand] }
20
21 console.log(test.fn()); // => test
22 console.log(test.fnShortHand()); // => test short hand
23
```

Object #8: ES6 Features - Property Value Shorthand & Destructuring

```
1
2  // ES6: Property Value Shorthand
3  let male = 'man';
4  let female = 'woman';
5  let test2 = {
6      male,
7      female,
8      // 'a' // => Throw: Unexpected token errors
9  }
10 console.log(test2); // => { male: 'man', female 'woman' }
11
12 // ES6: Destructuring
13 let capitals = {
14     Bangkok: 'Thailand',
15     Tokyo: 'Japan',
16     London: 'England'
17 }
18 let {Bangkok, Tokyo, London} = capitals;
19 // Equals to
20 // let Bangkok = capitals.Bangkok;
21 // let Tokyo = capitals.Tokyo;
22 // let London = capitals.London;
23 console.log(Bangkok, Tokyo, London); // => Thailand Japan England
24
```

Object #9: ES6 Features - Sprades Operators

```
1
2  // ES6: Spreades Operators ...
3  let sprade1 = {
4      a: 1,
5      b: 2,
6      c: 3
7  };
8  let sprade2 = {
9      ...sprade1,
10     c: 4,
11     d: 5,
12     e: 6
13 };
14 let sprade3 = {
15     c: 4,
16     d: 5,
17     e: 6,
18     ...sprade1
19 };
20 console.log(sprade2); // => { a: 1, b: 2, c: 4, d: 5, e: 6 }
21 console.log(sprade3); // => { c: 3, d: 5, e: 6, a: 1, b: 2 }
22
```

Exercise 1

```
1
2  let a = { 'th': 'thai', 'en': 'english' };
3  let b = { 'th': 'siam', 'jp': 'japan', 'cn': 'china' };
4  console.log({a,b});
5  console.log({...a,b});
6  console.log({...a,...b});
7
```

Object's Airbnb Style Guides

- Style Guide #1. Use literal syntax for object creation.

```
// bad
const item = new Object();

// good
const item = {};
```

Style Guide #2: Define All Property in One Place

```
function getKey(k) {
    return `a key named ${k}`;
}

// bad
const obj = {
    id: 5,
    name: 'San Francisco'
};
obj[getKey('enabled')] = true;

// good
const obj = {
    id: 5,
    name: 'San Francisco',
    [getKey('enabled')]: true
};
```

Style Guide #3: Use Object Method Shorthand

```
const atom = {
    value: 1,
    addValue: function(value) {
        return atom.value + value;
    }
};

// good
const atom = {
    value: 1,
    addValue(value) {
        return atom.value + value;
    }
};
```

Style Guide #4: Use Property Value Shorthand

```
const lukeSkywalker = 'Luke Skywalker';

// bad
const obj = {
    lukeSkywalker: lukeSkywalker
};

// good
const obj = {
    lukeSkywalker
};
```

Style Guide #5: Group your shorthand properties

```
const anakinSkywalker = 'Anakin Skywalker';
const lukeSkywalker = 'Luke Skywalker';

// bad
const obj = {
    episodeOne: 1,
    twoJediWalIntoACantina: 2,
    lukeSkywalker,
    episodeThree: 3,
    mayTheFourth: 4,
    anakinSkywalker
};

// good
const obj = {
    lukeSkywalker,
    anakinSkywalker,
    episodeOne: 1,
    twoJediWalIntoACantina: 2,
    episodeThree: 3,
    mayTheFourth: 4,

};
```

Style Guide #6: Only quote properties that are invalid identifiers

```
// bad
const bad = {
    'foo': 3,
    'bar': 4,
    'data-blah': 5
};

// good
const good = {
    foo: 3,
    bar: 4,
    'data-blah': 5
};
```

Style Guide #7: Use dot notation when accessing properties.

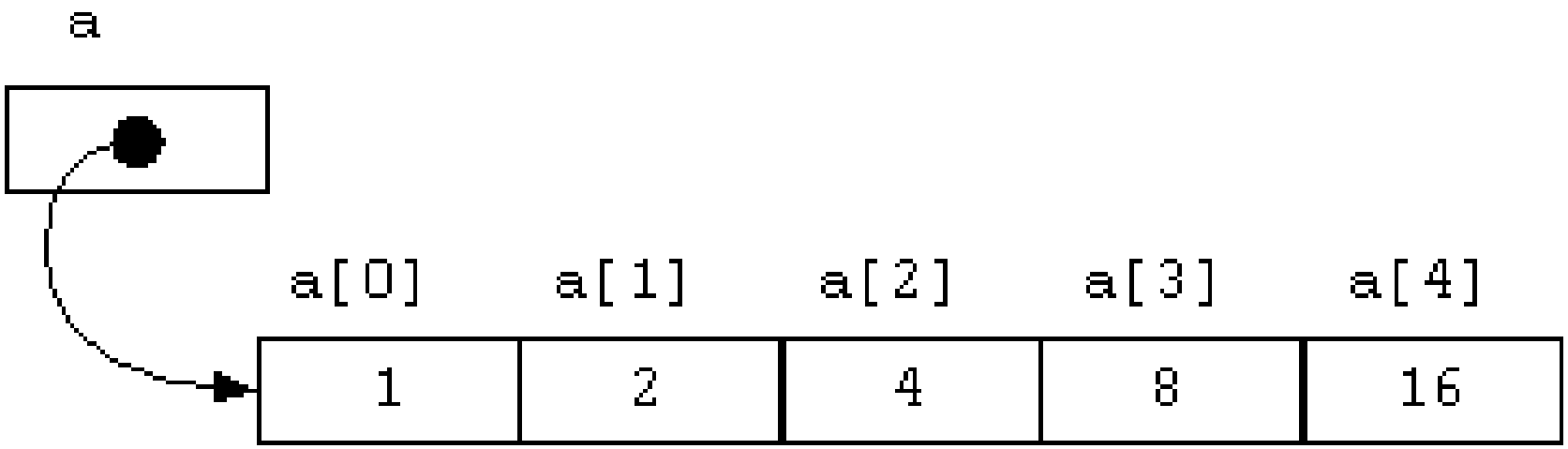
- Use bracket when accessing properties with a variable.

```
const lule = {
    jedi: true,
    age: 28
};

// bad
const isJedi = luke['jedi'];

// good
const isJedi = luke.jedi;
```

```
const luke = {
    jedi: true,
    age: 28
};

function getProp(prop) {
    return luke[prop];
}

const isJedi = getProp('jedi');
```

Array

- Array is a list data structure
- Sequence of objects



Array #1: Declaration, Update, Length

```
1
2  let arra = [];
3  let arr1 = [1,2,3];
4  let arr2 = [1,2,'abc'];
5  console.log(typeof(arr2[1])); // => number
6  console.log(typeof(arr2[2])); // => string
7  console.log(arr2); // => object, Array are objects!
8  console.log(arr2.length); // => 3, use .length to get its size
9
10 // Update array and add array at index.
11 arr2[0] = 10;
12 console.log(arr2); // => [10, 2, 'abc']
13 arr2[3] = 11;
14 console.log(arr2); // => [10, 2, 'abc', 11]
15
16 // Length = Largest Index + 1
17 let longarr = [];
18 longarr.push(0);
19 longarr[10000] = 0;
20 // console.log(longarr); // => [0, undefined x 9999 ..., 0]
21 console.log(longarr.length); // => 10001
22
```

Array #2: Delete, Sub-Array

```
1
2  // Delete element
3  let arr3 = [1,2,3,4,5];
4  delete arr3[2];
5  console.log(arr3); // => [1, 2, undefined, 4, 5]
6  console.log(arr3.length); // => 5
7
8  // How to actually remove its element? and insert it in between?
9  arr3.splice(2, 1); // splice({start index}, {amount of delete},
10                    // {element to insert ... });
11 console.log(arr3); // => [1,2,3,4,5]
12 arr3.splice(2, 1, 6, 7, 8);
13 console.log(arr3); // => [1,2,6,7,8,5]
14
15 // How to sub-array?
16 let arr4 = [1, 2, 3, 4, 5];
17 let arr5 = arr4.slice(1,3); // slide({start index},
18                             // {end index (not inclusive)})
19 console.log(arr4); // => [1,2,3,4,5]
20 console.log(arr5); // => [2,3]
21
```

Array #3: Push, Pop, Unshift, Shift

```
1
2  // Push element
3  let arr6 = [1, 2];
4  arr6.push(3);
5  arr6.push(null);
6  console.log(arr6); // => [1,2,3,null]
7
8  // Pop element
9  console.log(arr6.pop()); // => null
10 console.log(arr6.pop()); // => [1,2,3]
11
12 // Unshift element
13 arr6.unshift(0); // Push the first item of an array
14 console.log(arr6); // => [0, 1, 2, 3]
15
16 // Shift element
17 arr6.shift(); // Remove the first item of an array
18 console.log(arr6); // => [1,2,3]
19
```

Array #4: Array.isArray, [].indexOf

```
1
2  // How to reconize array
3  console.log(typeof([])); // => object
4  console.log(typeof({})); // => object
5  console.log(Array.isArray([])); // => true
6  console.log(Array.isArray({})); // => false
7
8  // Find element index in array
9  let arr7 = [1,2,3,4];
10 console.log(arr7.indexOf(2)); // => 1
11
12 // String is a kind of array
13 let str = 'Hello World!';
14 console.log(str.indexOf('H')); // => 0
15
```

Array #5: Loop

```
1
2  // Lopps
3  arr7.c = 999; // Assign element in array by accident...
4  for (let key in arr7) {// Don't use! Something it can have error occurs!
5                         // If array is not pure.
6      console.log(key, arr7[key]); // => 0 1
7                                   // => 1 2
8                                   // => 2 3
9                                   // => 3 4
10                                  // => c 999
11 }
12
13 for (let key = 0; key < arr7.length; key++) { // Safer!
14     console.log(key, arr7[key]); // => 0 1
15                                  // => 1 2
16                                  // => 2 3
17                                  // => 3 4
18                                  // => c 999
19 }
20
```

Exercise

- Write down a function to be able to add a number to all numbers in an array.

```
function addNumberToArray(array, number) {



}

console.log(addNumberToArray([1,2,3], 5)); // [6,7,8]
console.log(addNumberToArray([1,2,3], 1)); // [2,3,4]
```

Array #6: Sort, Reverse

```
1
2   // Sort
3   let arr8 = [2,3,1,4];
4   arr8.sort();
5   console.log(arr8); // => [1,2,3,4]
6
7   arr8.sort(function(a,b) { return b - a });
8   console.log(arr8); // => [4,3,2,1]
9
10  // Reverse Array
11  let arr9 = [0,2,1,'y','a','b','z'];
12  arr9.sort();
13  console.log(arr9); // => [0, 1, 2, 'a', 'b', 'y', 'z']
14  arr9.reverse();
15  console.log(arr9); // => ['z', 'y', 'b', 'a', 2, 1, 0]
16
```

Array #7: Array <-> String, 2D and 3D Array

```
1
2   // Transform to String
3   console.log(arr9.join('|')); // => 'z|y|b|a|2|1|0'
4
5   // Transform String to Array
6   let str1 = 'hello world! every body';
7   console.log(str1.split(" ")); // => [ 'hello', 'world!', 'every', 'body']
8
9   // 2D, 3D Array...
10  let array2d = [[1,2,3], [4,5,6], [7,8,9]];
11  let array3d = [[[1,2,3], [4,5,6], [7,8,9]], [[0,0,0], [0,0,0], [0,0,0]]];
12  array2d[1] = [10,11,12];
13  array2d[1][1] = 13;
14  console.log(array2d); // [[1, 2, 3], [10, 13, 12], [7, 8, 9]]
15
```

Array #8: [].forEach

```
1
2   // Foreach: Iterate over the member in Array
3   array2d.forEach((item, key) => {
4       console.log(item, key); // => [1, 2, 3] 0
5                               // => [10, 13, 12] 1
6                               // => [7, 8, 9] 2
7   });
8   array2d.forEach((item) => {
9       console.log(item); // => [1, 2, 3]
10                         // => [10, 13, 12]
11                         // => [7, 8, 9]
12  });
13
```

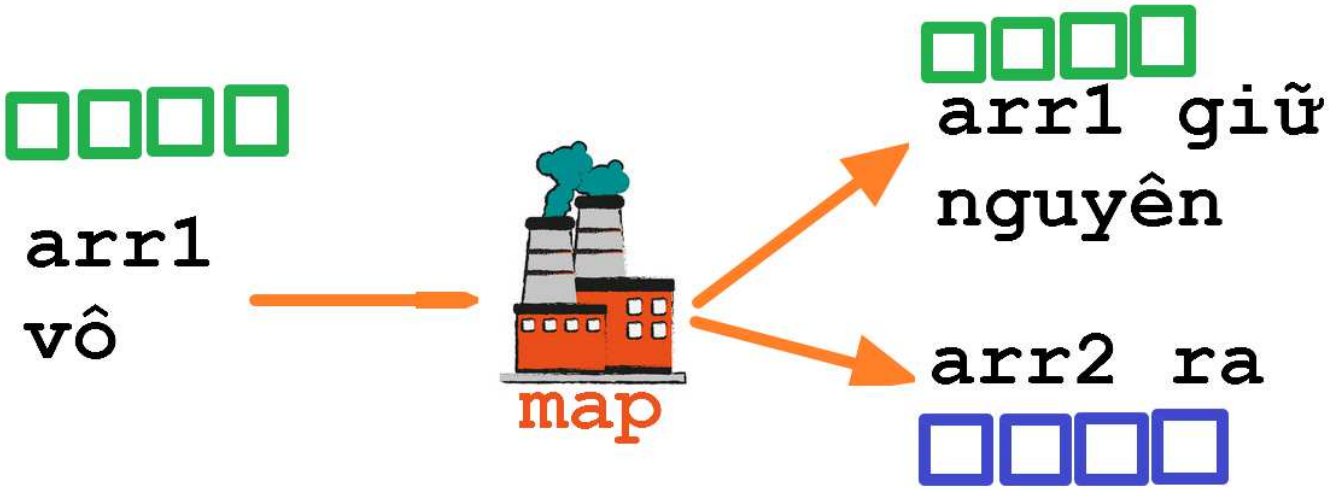
Array #9: ES6 Features: Array Spreads ..., Array.from, Array.of

```
1
2  // ES6: Array Spreads
3  let s1 = [1,2,3];
4  let s2 = [4,5,6];
5  let s3 = [...s1, ...s2];
6  let s4 = [...s2, 10, ...s1];
7  console.log(s3); // => [1, 2, 3, 4, 5, 6]
8  console.log(s4); // => [4, 5, 6, 10, 1, 2, 3]
9
10 // ES6: Array.from , Use for copy array, or convert array-like
11 // object into an array
12 console.log(Array.from(['a','b','c'])); // => ['a','b','c']
13 console.log(Array.from('abc')); // => ['a', 'b', 'c']
14
15 // ES6: Array.of, Create an array with given arguments as elements.
16 console.log(Array.of(1)); // => [1]
17 console.log(Array.of(1,2,3)); // => [1,2,3]
18 console.log(Array.of([1,2],3)); // => [[1,2],3]
19
```

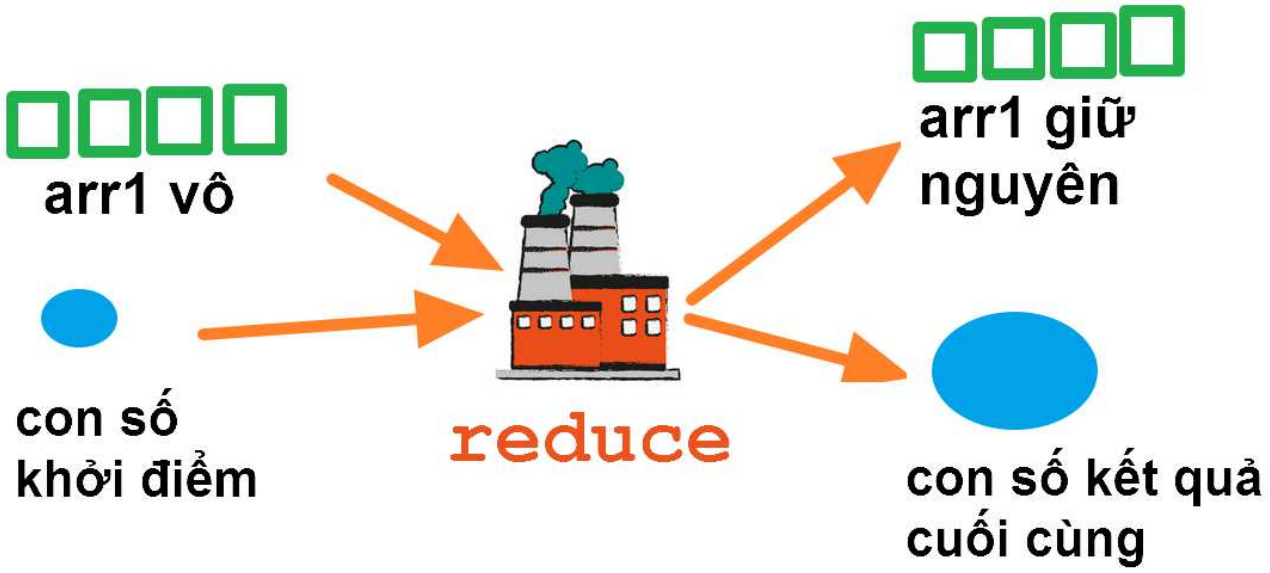
Array #10: ES6 Features: [].fill, [].find, [].findIndex, [].map

```
1
2  // ES6: [].fill({input element}, {start index},
3  //       {end index not inclusive}), Fill array with value
4  console.log(new Array(10).fill(1)); // => [1,1,1,1,1,1,1,1,1,1]
5  let arrFill = [1,2,3,4,5,6,7,8,9,10];
6  console.log(arrFill('a',3,5)); // => [1,2,3,'a','a',6,7,8,9,10]
7
8  // ES6: [].find(condition function), return element of array that
9  //       match the condition
10 //      [].findIndex(condition function), return index of array that
11 //       match the condition
12 console.log([1,2,10,20,30].find(ele => ele > 2)); // => 10
13 console.log([1,2,10,20,30].findIndex(ele => ele > 2)); // => 2
14
15 // ES6: [].map, transform, arrays
16 let source = [1,2,3,4,5];
17 console.log(source.map(x => x*x)); // => [1, 4, 9, 16, 25]
18 console.log(source.map(x => x+10)); // => [11, 12, 13, 14, 15]
19
```

Map

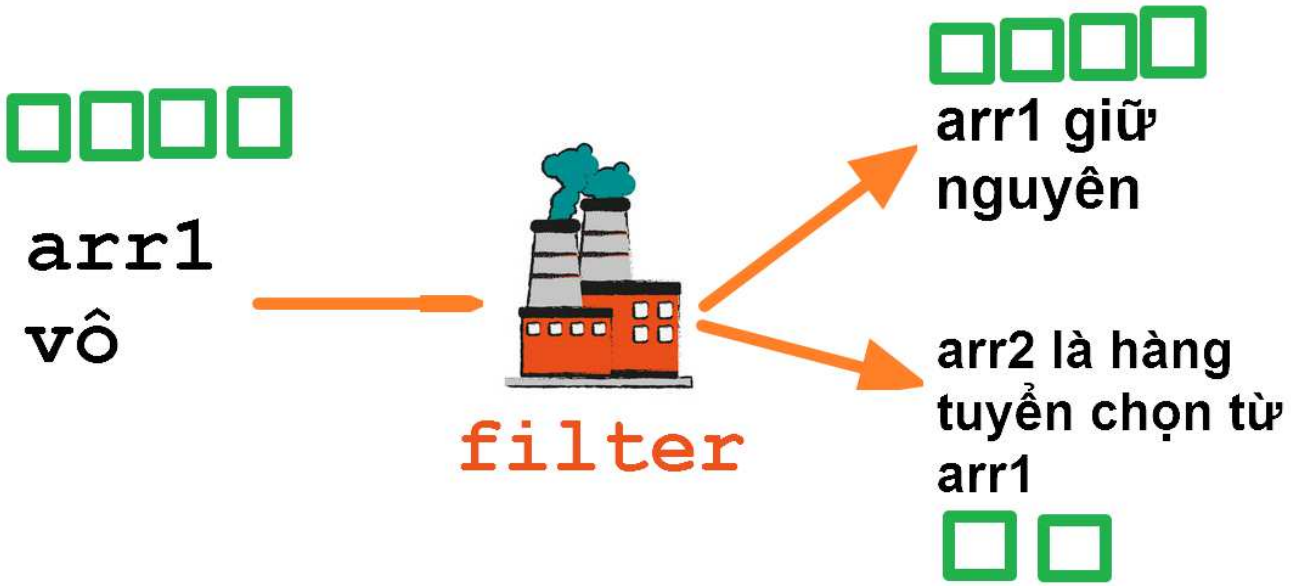


Reduce



HIENLTH

Filter


arr1
vô
filter
arr1 giữ nguyên
arr2 là hàng tuyển chọn từ arr1

Array #11: ES6 Features: [].reduce, [].filter

```
1
2   // ES6: [].reduce, reduce arrays to values
3   source = [1,2,3,4,5];
4   let sum = source.reduce((memo, item, index) => {
5       return memo + item;
6   });
7   let product = source.reduce((memo, item, index) => memo*item);
8   let concat = source.reduce((memo, item, index) => ''+memo+item);
9   console.log(sum); // => 15
10  console.log(product); // => 120
11  console.log(concat); // => 12345
12
13  // ES6: [].filter, filter some element in arrays
14  source = [1,2,3,4,5];
15  console.log(source.filter(x => x>3)); // => [4, 5]
16
```

Array #12: ES6 Features: Destructuring

```
1
2   // Destructuring
3   let arr11 = [1,2,3,4,5];
4   let [a,b,c] = arr11;
5   console.log(arr11, a, b, c); // => [1,2,3,4,5] 1 2 3
6
```

Exercise 3

- Do the same with Exercise, but using array.map only!

```
function addNumberToArray(array, number) {


}

console.log(addNumberToArray([1,2,3], 5)); // [6,7,8]
console.log(addNumberToArray([1,2,3], 1)); // [2,3,4]
```

Array's Airbnb Style Guides

- Style Guide #1. Use literal syntax for array creation.

```
// bad
const items = new Array();

// good
const items = [];
```

Style Guide #2. Use Array.push()

- Use Array.push() instead of direct assignment to add items to an array.

```
const someStack = [];

// bad
someStack[someStack.length] = 'abracadabra';

// good
someStack.push('abracadabra');
```

Style Guide #3. Use array spreads ... to copy arrays.

```
// bad
const len = items.length;
const itemCopy = [];
let i;

for (i = 0; i < len; i++) {
    itemCopy[i] = item[i];
}

// good
const itemsCopy = [...items];
```

Style Guide #4-1 Use return statements

- Use return statements in array method callbacks. It's ok to omit the return if the function body consists of a single statement

```
// bad
const flat = {};
[[0, 1], [2, 3], [4, 5]].reduce((memo, item, index) => {
    const flatten = memo.concat(item);
    flat[index] = flatten;
});

// good
const flat = {};
[[0, 1], [2, 3], [4, 5]].reduce((memo, item, index) => {
    const flatten = memo.concat(item);
    flat[index] = flatten;
    return flatten;
});
```

Style Guide #4-2 Use return statements

- Use return statements in array method callbacks. It's ok to omit the return if the function body consists of a single statement

```
// good
[1,2,3].map((x) => {
    const y = x + 1;
    return x * y;
});

// good
[1,2,3].map(x => x + 1);
```

```
// good
[1,2,3].map(x => x + 1);

// bad
inbox.filter((msg) => {
    const { subject, author } = msg;
    if (subject === 'Mockingbird') {
        return author === 'Harper Lee'
    } else {
        return false;
    }

    return false;
});
// good
inbox.filter((msg) => {
    const { subject, author } = msg;
    if (subject === 'Mockingbird') {
        return author === 'Harper Lee'
    }

    return false;
});
```

String

- String #1: Declaration, Concatenation, NewLine

```
1
2   // Declaration
3   const firstName = 'Anh';
4   const lastName = 'Huynh';
5   const positionName = 'culy';
6
7   // Concatenation String
8   const fullName = firstName + ' ' + lastName;
9
10  // Newline
11  const fullNameWithPosition = fullName + '\n' + position;
12
13  console.log(fullName); // => Anh Huynh
14  console.log(fullNameWithPosition); // => Anh Huynh
15                                     // culy
16
```

String #2: ES6 Feature - Template Strings

```
1
2  // ES6: Template Strings
3  const fullName2 = `${firstName} ${lastName}`;
4  const fullNameWithPosition2 = `${firstName}\n${position}`;
5  const fullNameWithPosition3 = `${fullName2}
6  ${position}`;
7
8  console.log(fullName2); // => Anh Huynh
9  console.log(fullNameWithPosition2);  // => Anh Huynh
10                                      //    culy
11 console.log(fullNameWithPosition3);  // => Anh Huynh
12                                      //    culy
13
```

String #3: Number <-> Strings

```
1
2   // Convert Number to String
3   const PI = 3.1428;
4   console.log(`PI = ${PI.toFixed(2)}`); // => PI = 3.14
5   // Only 2 decimal places
6
7   // Convert from String to Number
8   let sum = parseInt('3') + parseFloat('3.14');
9   console.log('sum'); // => 6.140000000001
10  console.log(sum.toFixed(2)); // => 6.14
11
```

String #4: IndexOf, Search, Replace

```
1
2   // indexOf, lastIndexOf
3   console.log('this is an apple'.indexOf('this')); // => 0
4   console.log('this is an apple'.indexOf('is')); // => 2 (at this)
5   console.log('this is an apple'.indexOf('mac')); // => -1
6   console.log('this is an apple'.indexOf('is')); // => 5 (at is)
7
8   // Search
9   console.log('this is an apple'.search(/a/)); // => 8
10
11  // Replace
12  console.log('this is an apple'.replace(/a/g,'b')); // => this is bn bpple
13
```

String #5: IndexOf, Search, Replace, Compare

```
1
2  // indexOf, lastIndexOf
3  console.log('this is an apple'.indexOf('this')); // => 0
4  console.log('this is an apple'.indexOf('is')); // => 2 (at this)
5  console.log('this is an apple'.indexOf('mac')); // => -1
6  console.log('this is an apple'.indexOf('is')); // => 5 (at is)
7
8  // Search
9  console.log('this is an apple'.search(/a/)); // => 8
10
11 // Replace
12 console.log('this is an apple'.replace(/a/g,'b')); // => this is bn bpple
13
14 // Compare
15 console.log('ABC' === 'ABC'); // true
16 console.log('ABC' === 'abc'); // false
17 console.log('ABC' == 'ABC'); // true
18 console.log('ABC' == 'abc'); // false
19 console.log('ABC' != 'ABC'); // false
20 console.log('abc' < 'z'); // true
21
```

String #6: charAt, slice, toLowerCase, toUpperCase, trim, includes

```
1
2  // char at
3  console.log('this is an apple'.charAt(0)); // => t
4  console.log('this is an apple'.charAt(1)); // => h
5  console.log('this is an apple'[2]); // => i
6
7  // slice(), toLowerCase(), toUpperCase(), trim()
8  console.log('this is an apple'.slice(5,8)); // => is
9  console.log('this is an apple'.toLocaleLowerCase()); // => this is an apple
10 console.log('this is an apple'.toUpperCase()); // => THIS IS AN APPLe
11 console.log('this is an apple '.trim()); // => this is an apple
12
13 // ES6: Text Includes
14 console.log('this is an apple '.includes('apple')); // => true
15 console.log('this is an apple '.includes('mac')); // => false
16
```

String #7: startsWith, endsWith, Repeats

```
1
2  // ES6: Text startsWith
3  console.log('this is an apple'.startsWith('this')); // => true
4  console.log('this is an apple'.startsWith('is')); // => false
5
6  // ES6: Text endsWith
7  console.log('this is an apple'.endsWith('apple')); // => true
8  console.log('this is an apple'.endsWith('an')); // => false
9
10 // ES6: Text repeats
11 console.log('x'.repeat(0)); // =>
12 console.log('x'.repeat(1)); // => x
13 console.log('x'.repeat(10)); // => xxxxxxxxxx
14
```

Exercise

- Write function that count word size case-insensitively.
- Input: "Hello world hello hello earth earth" (Not limited to these word, it can be any words)
- Output: Object{hello : 3, world : 1, earth : 2 }

String's Airbnb Style Guides

- Style Guide #1. Use single quote ' ' for Strings

```
// bad
const name = "Capt. Janeway"

// bad - template literals should contain interpolation or newlines
const name = `Capt. Janeway`;

// good
const name = 'Capt. Janeway';
```

Style Guide #2

- Strings that cause the line to go over 100 characters should not be

```
// bad
const errorMessage = 'This is a super long error that was thrown because \
of Batman. When you stop to think about how Baman had anything to do \
with this, you would get nowhere \
fast.';

// bad
const errorMessage = 'This is a super long error that was thrown because ' +
'of Batman. When you stop to think about how Baman had anything to do ' +
'with this, you would get nowhere fast.';

// good
const errorMessage = 'This is a super long error that was thrown because of Batman. When you stop
```

Style Guide #3

- When programmatically building up strings, use template strings instead of concatenation

```
// bad
function sayHi(name) {
    return 'How are you, ' + name + '?';
}

// bad
function sayHi(name) {
    return ['How are you, ', name, '?'].join();
}

// bad
function sayHi(name) {
    return `How are you, ${name}?`;
}

// good
function sayHi(name) {
    return `How are you, ${name}?`;
}
```

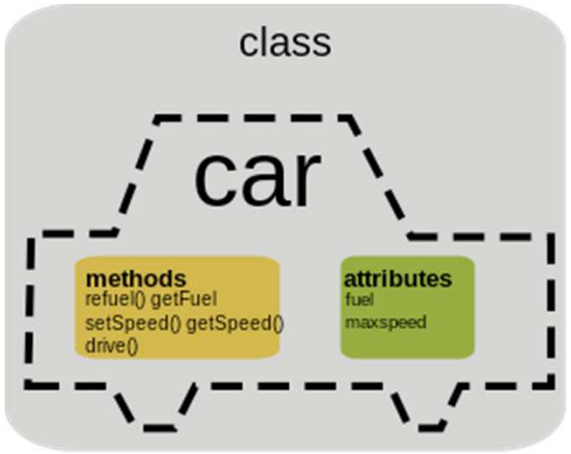
Style Guide #4

- Do not unnecessarily escape characters in strings.

```
// bad
const foo = '\'this\' \i\s \"quoted\"';

// good
const foo = '\'this\' \i "quoted"';
const foo = `'this' is "quoted"`;
```

Class

- Programming Paradigm that Encapsulate data and methods based on the concept of real world objects.
- JavaScript never have Class concept before. They just added in ES6.
- By adding the Class, it make the object-oriented programming in JavaScript a lot easier and nicer.



Class #1: ES6 Feature - Declaration, Instantiate

```
1
2  // ES6: syntax sugar improve classes
3  class Cat {
4      constructor(name, color) {
5          this.name = name;
6          this.color = color;
7      }
8
9      present() {
10         return `${this.name} is ${this.color}`;
11     }
12 }
13
14 var cat = new Cat('Mici', 'dark brown');
15 console.log(cat.present()); // => Mici is dark brown
16
```

Class #2: ES6 Feature - Extend

```
1
2  // ES6: Extend
3  class Tabby extends Cat {
4      constructor(name) {
5          super(name, ['Orange', 'White']);
6      }
7
8      present() {
9          return `${super.present()} and she's cool`;
10     }
11 }
12
13 let cat = mew Tabby('Friskies Cat', 'Friskies Cat');
14 console.log(cat.present()); // Friskies Cat is orange
15                             // White and she's cool
16
```

Class #3: ES6 Feature - Getter/Setter

```
1
2  // ES6: getters/settes are great
3  class Cat {
4      constructor(name, gender) {
5          this._name = name;
6          this.gender = gender;
7      }
8
9      // name is like a property, but calculatable
10     get name() {
11         if(this.gender == 'female') {
12             return `Ms. ${this._name}`;
13         } else {
14             return `Mr. ${this._name}`;
15         }
16     };
17
18     set name (newName) {
19         if (newName) {
20             this._name = newName;
21         } else {
22             console.log('I need a name');
23         }
24     }
25 }
```

```
27 let cat = new Cat('Mici', 'female');
28 console.log(cat.name) // Ms. Mixi
29
30 cat.name = null; // I need a name
31
32 cat.name = 'John';
33 cat.gender = 'male';
34 console.log(cat.name);
35
```

Class's Airbnb Style Guides

- Style Guide #1. Methods can return this, to enable method chaining.

```
1
2   // bad
3   Jedi.prototype.jump = function() {
4       this.jumpping = true;
5       return true;
6   }
7
8   Jedi.prototype.height = function(height) {
9       this.height = height;
10  }
11
12  const luke = new Jedi();
13  luke.jump(); // => true
14  luke.setHeight(20); // => undefined
15
```

```
16  // Good
17  class Jedi {
18      jump() {
19          this.jumpping = true;
20      }
21
22      setHeight(height) {
23          this.height = height;
24          return this;
25      }
26  }
27
28  const luke = new Jedi();
29  luke.jump().setHeight(20);
30
```

Style Guide #2

- Classes have a default constructor if one is not specified. An empty constructor function or one that just delegates to a parent class is unnecessary.

```
// bad
class Jedi {
    constructor() {};

    getName() {
        return this.name;
    }
}

// bad
class Rey extends Jedi {
    constructor(...args) {
        super(...args);
    }
}
```

```
// good
class Rey extends Jedi {
    constructor(...args) {
        super(...args);
        this.name = 'Key';
    }
}
```

HomeWork

Let’s say: '(', '{', '[' are called “openers."

')', '}', ']' are called "closers."

Write an efficient function that tells us whether or not an input string's openers and closers are properly nested.

Examples:

"{ [ ] ( ) }" should return true

"{ [( ]) }" should return false

"{ [}" should return false

Thank you!