JAVASCRIPT

Biên soạn: **Phạm Hữu Bảo Chung**

MỤC LỤC

PHẦN A. 8.5 ĐIỂM

I. JavaScript Algorithms and Data Structures Certification

1. Basic JavaScript

Câu 1.1. [Comment]

```
// hoặc /* */
```

Câu 1.2. [Khai báo và khởi tạo biến]

```
var a = 9;
```

Câu 1.3. [+, -, *, /, %]

```
a = 10 + 20;
```

Câu 1.4. [++, --, +=, -=, *=, /=]

```
a++;
a += 5;
```

Câu 1.5. [Xâu – Lưu ý: Immutable]

```
var str = “Hello world!”
str = “Hello \”worldQ!\””;
str = “Hello ‘world!’”
str = “FirstLine\n\t\\SecondLine\nThirdLine”;
str = “Hello ” + “world”;
str += “!”;
str = “Hello  ” + a + “ worlds!”
var len = str.length;

// [Tìm phần tử đầu]
var firstLetter = str[0];

// [Tìm phần tử thứ ba]
var thirdLetter = str[2];

// [Tìm phần tử cuối]
var lastLetter = str[str.length - 1];

// [Tìm phần tử thứ ba từ dưới lên]
var thirdToLastLetter = str[str.length - 3];

// {Bài tập – Nối danh từ/ tính từ/ động từ/ trạng từ}
function wordBlanks(myNoun, myAdjective, myVerb, myAdverb) {
    var result = myAdjective + " " + myNoun + " " + myVerb + " " + myAdverb + ".";
    return result;
}
wordBlanks("dog", "big", "ran", "quickly");
```

Câu 1.6. [Mảng]

```
var arr = [1, 2, 3];
var item = arr[0];
arr[1] = 2;
var arr2 = [[1, 2], [3, 4]];
arr2[0][0] = 2;

// [push(), pop(), shift(), unshift()]
arr.push(4);                    // arr = [1, 2, 3, 4]
var firstItem = arr.pop();      // arr = [1, 2, 3]
var lastItem = arr.shift();     // arr = [2, 3]
arr.unshift(1);                 // arr = [1, 2, 3]
```

Câu 1.7. [Hàm]

```
function f (a, b) {
        console.log(a + b);
        return a - b;
}
```

Câu 1.8. [If]

```
if (condition) {
        statements;
}

// [==, ===, !=, !==]
1 == "1";                               // true
1 === "1"                               // false
1 != "1";                       // false
1 !== "1";                              // true

// [>, >=, <, <=]
'2' > 1                                 // true

// {Bài tập}
function testSize(num) {
        if (num < 5)            return "Tiny";
        else if (num < 10)      return "Small";
        else if (num < 15)      return "Medium";
        else if (num < 20)      return "Large";
        else                    return "Huge";
}
```

Câu 1.9. [Switch]

```
switch (num) {
        case value1:
                statement1;
                break;
        case value2:
                statement2;
                break;
        ……
        default:
                defaultStatement;
                break;
        }

// {Bài tập – Trò chơi Counting Card}
var count = 0;
function cc(card) {
        switch(card) {
                case 2:
                case 3:
                case 4:
                case 5:
                case 6:
                        count++;
                        break;
                case 7:
```

```
                case 8:
                case 9:
                        break;
                case 10:
                case 'J':
                case 'Q':
                case 'K':
                case 'A':
                        count--;
                        break;
        }

        if (count > 0)
                return count + " Bet";
        else
                return count + " Hold";
}
cc(2); cc(3); cc(7); cc('K'); cc('A');
```

Câu 1.10. [Đối tượng]

```
var object = {
        prop1: “val1”,
        “prop 2”: “val2”
        prop3: {
                prop3a: “val3a”,
                prop3b: “val3b”
        }
}
var data = object.prop1;
var data2 = object[“prop 2”];
var i = “prop 2”;
var data3 = object[i];
object.prop1 = “Hello World!”;
object.prop4 = “val4”;
delete object.prop4;
object.hasOwnProperty(“prop 2”);
```

Câu 1.11. [Vòng lặp]

```
for ([initialization]; [condition]; [final-expression]) {
        [statements];
}

[initialization];
while ([condition]) {
        [statements];
        [final-expression];
}

[initialization];
do {
        [statements];
        [final-expression];
} while ([condition]);
```

```
// {Bài tập – Truy cứu thông tin}
var contacts = [
  {
    "firstName": "Akira",
    "lastName": "Laine",
    "number": "0543236543",
    "likes": ["Pizza", "Coding", "Brownie Points"]
  },
  {
    "firstName": "Harry",
    "lastName": "Potter",
    "number": "0994372684",
    "likes": ["Hogwarts", "Magic", "Hagrid"]
  },
  {
    "firstName": "Sherlock",
    "lastName": "Holmes",
    "number": "0487345643",
    "likes": ["Intriguing Cases", "Violin"]
  },
  {
    "firstName": "Kristian",
    "lastName": "Vos",
    "number": "unknown",
    "likes": ["JavaScript", "Gaming", "Foxes"]
  }
];

function lookUpProfile(name, prop){
  for(var i = 0; i < contacts.length; i++) {
    if(contacts[i].firstName === name) {
      if(contacts[i].hasOwnProperty(prop)) {
        return contacts[i][prop];
      } else {
        return "No such property";
      }
    }
  }

  return "No such contact";
}
```

Câu 1.12. [Số ngẫu nhiên]

```
Math.random();                    // Sinh một số trong nửa khoảng [0, 1)
Math.floor(Math.random() * 20);
Math.floor(Math.random() * (max - min + 1)) + min;
```

Câu 1.13. [Hàm parseInt]

```
var a = parseInt("007");          // a = 7
var a = parseInt("11", 2);        // a = 3
```

2. ES6

Câu 2.1. [let]

```
'use strict';
let printNumTwo;
for (let i = 0; i < 3; i++) {
    if (i === 2) {
        printNumTwo = function() {
            return i;
        };
    }
}
console.log(printNumTwo());                    // returns 2
```

Câu 2.2. [const]

```
// [VD1.]
"use strict"
const FAV_PET = "Cats";
FAV_PET = "Dogs";                              // returns error
// [VD2.]
"use strict";
const s = [5, 6, 7];
s = [1, 2, 3];                                 // throws error
s[2] = 45;                                     // works
console.log(s);                                // returns [5, 6, 45]
```

Câu 2.3. [Object.freeze()]

```
let obj = {
    name: "FreeCodeCamp",
    review: "Awesome"
};
Object.freeze(obj);                            // Đối tượng không thể thay đổi
```

Câu 2.4. [Hàm =>]

```
// [VD1.]
const myFunc = function() {
    const myVar = "value";
    return myVar;
};

// [VD2.]
const myFunc = () => {
    const myVar = "value";
    return myVar;
};

//[VD3.]
const myFunc = () => "value";

//[VD4.]
const doubler = (item) => item * 2;

//[VD5. Hàm => thường được sử dụng với các hàm map(), filter() và reduce()]
FBPosts.filter((post) => post.thumbnail !== null && post.shares > 100 && post.likes > 500);

//{Lưu ý – Ta có thể kiểm tra một số có phải Integer không qua điều kiện (num % parseInt(num) === 0) hoặc hàm Number.isInteger() }

//[VD6.]
```

```
const getMousePosition = (x, y) => ({x, y});          // Thay vì {x: x, y: y}
```

Câu 2.5. [Tham số mặc định]

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

Câu 2.6. [Rest operators]

```
function howMany(...args) {
    return "You have passed " + args.length + " arguments.";
}
console.log(howMany(0, 1, 2));                        // 3 tham số
console.log(howMany("string", null, [1, 2, 3], { })); // 4 tham số
```

Câu 2.7. [Spread operators]

```
const arr = [6, 89, 3, 45];
const maximus = Math.max(...arr);                     // Trả về 89
```

// {Lưu ý}

- Xét arr = [1, 2, 3, 4, 5].
- Khi đó, ...arr sẽ tương đương với dãy: 1, 2, 3, 4, 5.
- Có thể hiểu như sau: ...arr tạo ra một xâu các phần tử của mảng (được ngăn cách bởi các dấu phẩy) và dán nó vào vị trí cần dùng cú pháp này (như tham số các hàm hay vị trí giữa hai dấu [] khi khai báo mảng).
- Chẳng hạn: arr2 = [...arr]

Câu 2.8. [Gán nhãn đồng loạt]

// [VD1.]

```
var voxel = {x: 3.6, y: 7.4, z: 6.54 };
const { x, y, z } = voxel;                            // x = 3.6, y = 7.4, z = 6.54
```

// [VD2.]

```
const { x : a, y : b, z : c } = voxel                 // a = 3.6, b = 7.4, c = 6.54
```

// [VD3.]

```
const a = {
        start: { x: 5, y: 6},
        end: { x: 6, y: -9 }
};
const { start : { x: startX, y: startY }} = a;
console.log(startX, startY);                          // 5, 6
```

// [VD4.]

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

// {Lưu ý - Ta có thể tính độ dài một xâu dựa trên cú pháp: const {length: len} = str; do length là một "thuộc tính" của xâu.}

// {Lưu ý: Để đổi chỗ 2 số, ta có thể viết [a, b] = [b, a].}

// [VD5.]

```
const [a, b, ...arr] = [1, 2, 3, 4, 5, 7];
console.log(a, b);                                    // 1, 2
console.log(arr);                                     // [3, 4, 5, 7]
```

// [VD6.]

```
const profileUpdate = (profileData) => {
    const { name, age, nationality, location } = profileData;
    // do something with these variables
```

```
}
```

Câu 2.10. [Xâu nguyên thủy]

```
const person = {
    name: "Zodiac Hasbro",
    age: 56
};
const greeting = `Hello, my name is ${person.name}!
I am ${person.age} years old.`;
console.log(greeting);

//{Lưu ý}
```

- Sử dụng ` (dưới nút Esc) thay vì ', ".
- Có thể được khởi tạo trên nhiều hàng.
- Sử dụng ${variable} thay vì +.

Câu 2.11. [Khai báo hàm]

```
const person = {
    name: "Taylor",
    sayHello() {                          // Thay vì sayHello: function()
        return `Hello! My name is ${this.name}.`;
    }
};
```

Câu 2.12. [class]

```
class SpaceShuttle {
    constructor(targetPlanet){
        this.targetPlanet = targetPlanet;
    }
}
const zeus = new SpaceShuttle('Jupiter');
```

Câu 2.13. [getter và setter]

```
class Book {
    constructor(author) {
        this._author = author;
    }

    get writer(){
        return this._author;
    }

    set writer(updatedAuthor){
        this._author = updatedAuthor;
    }
}
const lol = new Book('anonymous');
console.log(lol.writer); lol.writer = 'wut';
console.log(lol.writer);
```

Câu 2.14. [import và export]

```
// [VD1.]
import { countItems } from "math_array_functions"

// [VD2.]
import { capitalizeString } from "string_functions"
"use strict";
capitalizeString("hello!");

//[VD3.]
```

```
const capitalizeString = (string) => {
        return string.charAt(0).toUpperCase() + string.slice(1);
}
export { capitalizeString }          //How to export functions.
export const foo = "bar";            //How to export variables.
```

//[VD4.]

```
import * as myMathModule from "math_functions";
myMathModule.add(2, 3);
myMathModule.subtract(5, 3);
```

TỔNG KẾT

- **[let]** được sử dụng thay var để tránh việc khai báo các biến cùng tên.
  VD: let x = 1;
- **[use strict]** được sử dụng như throw...catch ở Java.
  VD: "use strict";
- **[const]** được sử dụng để khai báo hằng số.
  VD: const x = 1;
- **[Object.freeze()]** được sử dụng để đảm bảo đối tượng và các thuộc tính của nó là không thể thay đổi. (Khi sử dụng var, đối tượng không được phép khởi tạo lại, tuy nhiên, ta vẫn có quyền thêm, xóa và sửa các thuộc tính của nó).
  VD: Object.freeze(A);
- **[Arrow functions]** được sử dụng để định nghĩa các hàm thông qua ký hiệu =>
  VD: const funcName = (x, y) => x + y;
- **[Default parameters]** được sử dụng để khai báo giá trị mặc định cho một tham số nếu nó không được truyền vào.
  VD: function funcName(x = 1){doSomething;}
- **[...]** được sử dụng để chuyển mảng thành một dãy các phần tử, thường được dùng để copy mảng này sang mảng khác.
  VD: let newA = [...A];
- **[Destructuring Assignment]** được sử dụng để giảm bớt số câu lệnh khởi tạo khi trích xuất thông tin từ đối tượng.
  VD:
  var A = {A1: 1, A2: 2, A3: 3};
  const { x, y, z } = A;
  [Template Literal] được sử dụng để thay thế phép + các xâu.
  VD:
  let A = {A1: 1, A2: 2};
  let str = `...${A.A1}...${A.A2}...`;
- **[Object Literal]** được sử dụng để xóa các cú pháp dư thừa.
  VD:
  const funcName = (x, y) => ({ x, y });
  thay vì
  const funcName = (x, y) => ({ x: x, y: y });
- **[Concise Declarative Functions]** được sử dụng để loại bỏ từ function khi xây dựng hàm như một thuộc tính của đối tượng.
  VD:
  const A = {A1: x, A2(){return `...${this.A1}...`;}};
  thay vì
  const A = {A1: x, A2: function(){return `...${this.A1}...`;}};
- **[class]** được sử dụng để thay thế việc tạo các constructor function.
  VD:
  class A{constructor(x){this.x = x;}}
  let objA = new A(1);
  thay vì

```
let A = function(x){this.x = x;}
let objA = new A(1);
```

- **[getters/setters]** được sử dụng tương tự ở Java.

```
VD:
class A{
        constructor(x) {this.x = x;}
        get F(){return this.x;}
        set F(x){this.x = x;}
}
A.F = 1;
```

- **[import/export/*]** là các chức năng để nhập/xuất code từ/sang file khác.

3. Regular Expression

Câu 3.1. [regex]

```
let testStr = "freeCodeCamp";
let testRegex = /Code/;
testRegex.test(testStr);
```

Câu 3.2. [Các ký hiệu thông dụng]

```
/yes|no/
/ignorecase/I                   ; đoán nhận cả iGnOreCAse
/hu./                           ; đoán nhận cả hug, huh, hut…
/b[aiu]g/                       ; đoán nhận cả bag, big, bug
/[a-e]at/                       ; đoán nhận (a -> e)at
/[a-z0-9]/i;                    ; đoán nhận  tất cả các chữ cái và chữ số
/[^aeiou]/gi
/a+/g                           ; abc -> [“a”], aabc -> [“aa”], abab -> [“a”, “a”]
                                ; sử dụng * để đoán nhận không hoặc nhiều.
/t[a-z]*?i/                     ; titanic -> ti (thay vì thông thường trả về titani)
/a{3,5}h/                       ; đoán nhận a xuất hiện 3 – 5 lần
/ha{3,}/                        ; đoán nhận a xuất hiện tối thiểu 3 lần
```

Câu 3.3. [.match()]

```
// [VD1.]
"Hello, World!".match(/Hello/);          // Returns ["Hello"]

// [VD2. Cờ g. Lưu ý, có thể sử dụng nhiều cờ trên một BTCQ.]
let repeatRegex = /Repeat/g;
testStr.match(repeatRegex);
// Returns ["Repeat", "Repeat", "Repeat"]
```

Câu 3.4. [^]

```
let firstString = "Ricky is first and can be found.";
let firstRegex = /^Ricky/;
firstRegex.test(firstString);            // Returns true
let notFirst = "You can't find Ricky now.";
firstRegex.test(notFirst);               // Returns false
```

Câu 3.5. [$]

```
let theEnding = "This is a never ending story";
let storyRegex = /story$/;
storyRegex.test(theEnding);              // Returns true
let noEnding = "Sometimes a story will have to end";
storyRegex.test(noEnding);               // Returns false
```

Câu 3.6. [\w = [A-Za-z0-9_]]

```
let longHand = /[A-Za-z0-9_]+/;
let shortHand = /\w+/;
let numbers = "42";
let varNames = "important_var";
longHand.test(numbers);                          // Returns true
```

```
shortHand.test(numbers);                // Returns true
longHand.test(varNames);                // Returns true
shortHand.test(varNames);               // Returns true
```

Câu 3.7. [\W = [^A-Za-z0-9_]]

```
let shortHand = /\W/;
let numbers = "42%";
let sentence = "Coding!";
numbers.match(shortHand);               // Returns ["%"]
sentence.match(shortHand);              // Returns ["!"]
// Lưu ý: [\d] = [0-9], [\D] = [^0-9], [\s] = Khoảng trắng
```

Câu 3.8. {Bài tập – Username}

```
let username = "JackOfAllTrades";
let userCheck = /[a-z]+[a-z]+\d*?/i;    // Change this line
let result = userCheck.test(username);
```

Câu 3.9. [?]

```
let american = "color";
let british = "colour";
let rainbowRegex= /colou?r/;
rainbowRegex.test(american);            // Returns true
rainbowRegex.test(british);             // Returns true
```

Câu 3.10. [?! và ?!]

```
// [VD1.]
let quit = "qu";
let noquit = "qt";
let quRegex= /q(?=u)/;
let qRegex = /q(?!u)/;
quit.match(quRegex);                    // Returns ["q"]
noquit.match(qRegex);                   // Returns ["q"]
//[VD2. Kiểm tra hai hoặc nhiều pattern trên cùng một câu ]
let password = "abc123";
let checkPass = /(?=\w{3,6})(?=\D*\d)/;
checkPass.test(password);               // Returns true

//{Bài tập}
let sampleWord = "astronaut";
let pwRegex = /(?=\w{5,})(?=\D*\d\d+)/;
let result = pwRegex.test(sampleWord);
```

Câu 3.11. [()]

```
let repeatStr = "regex regex";
let repeatRegex = /(\w+)\s\1/;
repeatRegex.test(repeatStr);        // Returns true
repeatStr.match(repeatRegex);       // Returns ["regex regex", "regex"]
```

//{Lưu ý}

- Xét ví dụ:

  ```
  let testString = "test test test ";
  let reRegex =/(test)\s\1/;
  let result = reRegex.test(testString);
  ```

  result sẽ chỉ khớp với test test do \1 ở đây tham chiếu đến đoạn bị nhóm đầu tiên, tức (test).

  Nói nôm na thì (test)\s\1 = test\stest
- Trong khi đó, ở ví dụ này:

  ```
  let testString = "test test test ";
  ```

```
let reRegex =/(test)(\s)\1\2\1/;
let result = reRegex.test(testString);
```

result sẽ khớp với toàn bộ test test test do có 2 đoạn text được gộp và \1 tham chiếu đến nhóm đầu tiên, tức (test), còn \2 tham chiếu đến nhóm thứ 2, tức \s

Nghĩa là: (test)(\s)\1\2\1 = test\stest\stest\s

//{Bài tập}

```
let reRegex = /^(\d+)\s\1\s\1$/;
```

Câu 3.12. [.replace()]

```
// [VD1.]
let wrongText = "The sky is silver.";
let silverRegex = /silver/;
wrongText.replace(silverRegex, "blue");

// [VD2.]
"Code Camp".replace(/(\w+)\s(\w+)/, '$2 $1');

// {Bài tập – Hàm trim() các khoảng trắng hai đầu của của một xâu}
let hello = " Hello, World! ";
let wsRegex = /^\s+|\s+$/g;                    // Change this line
let result = hello.replace(wsRegex, '');       // Change this line
```

TỔNG KẾT

- Sử dụng **/regex/.test("str")** để kiểm tra xem xâu str có chứa các xâu con khớp với regex hay không.
- **[|]** được sử dụng như OR.
  VD. let test = /x|y/;
- **[i]** được sử dụng để tránh phân biệt hoa/thường.
  VD. let test= /regEx/i;
- Sử dụng **"str".match(/regex/)** để trả về xâu con đầu tiên trong str khớp với regex. Sử dụng [g] nếu muốn khớp nhiều hơn một xâu con như vậy.
  VD. str.match(/regex/g);
- **[.]** được sử dụng để khớp với bất kỳ ký tự nào.
  VD: let test = /h./;
- **[[ ]]** được sử dụng để khớp một trong số các ký tự bên trong [ ]. Sử dụng [-] để chỉ một dãy các ký tự liên tiếp từ ký tự trước - đến ký tự sau a. Sử dụng [^] để khớp các ký tự khác các ký tự sau ^.
  VD:
  str.match(/[cb]at/g);
  str.match(/[a-zA-Z0-9]/);
  str.match(/[^a-zA-Z0-9]/);
- **[+]** được sử dụng để khớp một pattern xuất hiện liên tiếp ít nhất một lần trong câu. **[*]** được sử dụng để khớp một pattern xuất hiện liên tiếp ít nhất một lần hoặc không xuất hiện trong câu.
  VD:
  str.match(/[a-zA-Z]+/);
  str.match(/[0-9]*/);
- **[?]** được sử dụng để khớp xâu ngắn hơn khi có nhiều hơn một xâu khớp.
  VD. str.match(/h[a-z]*?o/);
- **[^]** được sử dụng để khớp xâu con ở đầu câu. **[$]** được sử dụng để khớp xâu con ở cuối xâu.
  VD.
  str.match(/^[A-Z]/);
  str.match(/.$/);
- **[\w]** được sử dụng để khớp các ký tự [A-Za-z0-9_]. [\W] khớp các ký tự khác các ký tự trong \w.
  VD. str.match(/\w+/);
- **[\d]** được sử dụng để khớp [0-9]. **[\D]** khớp các ký tự khác các ký tự trong \d.
  VD. str.match(/\d*/)
- **[\s]** được sử dụng để khớp các ký tự [\r\t\f\n\v]. [\S] khớp các ký tự khác các ký tự trong \S.

VD. str.match(/\s/);

- **[{ }]** được sử dụng để chỉ số lần lặp của một pattern.
  VD. str.match(/\w{3}/);
- **[?]** được sử dụng để chỉ một ký hiệu có thể xuất hiện hoặc không.
  VD. str.match(/colou?r/);
- **[?= và ?!]** được sử dụng để tìm kiếm nhiều pattern trên một xâu. ?= đảm bảo pattern có mặt trong xâu, nhưng không khớp nó. ?! đảm bảo pattern không có mặt trong xâu.
  VD.
  let password = "abc123";
  let checkPass = /(?=\w{3,6})(?=\D*\d)/;
  checkPass.test(password);
- **[( )]** được sử dụng khi pattern xuất hiện nhiều lần (không liên tiếp) trong câu. Sử dụng \số để tham chiếu đến thứ tự của pattern được lặp lại.
  VD. string.match(/(pattern1)(pattern2)\s\1\2/);
- **str.replace()** được sử dụng để thay thế một regex bằng một xâu.
  VD.
  str.replace(/A/, "B");
  str.replace(/(\w+)\s(\w+)/, "$2 $1");

4. <u>Debugging</u>

- Sử dụng [console] để in kết quả thực hiện của từng bước.
- Sử dụng [typeof] để kiểm tra kiểu dữ liệu.

5. <u>Basic Data Structures</u>

5.1. Mảng

- One-dimensional array
- Array.length
- Multi-dimensional array
- JavaScript arrays are zero-indexed
- Bracket notation

Câu 5.1.1. [push(), pop(), shift(), unshift()]

- arr.push(): Thêm phần tử vào cuối mảng.
  arr.unshift(): Thêm phần tử vào đầu mảng
  arr.pop(): Xóa phần tử cuối mảng, return phần tử này.
  arr.shift(): Xóa phần tẻ đầu mảng, return phần tử này.
- push() và unshift() đều nhận tham số. Ta có thể thêm nhiều phần tử vào mảng trong một lần gọi hàm.
- pop() và shift() không nhận tham số. Ta chỉ có thể xóa một phần tử trong mỗi lần gọi đến các hàm này.

Câu 5.1.2. [splice()]

- arr.splice() được dùng để xóa các phân tử liên tiếp bất kỳ.
- arr.splice() nhận 3 tham số.
  - Tham số 1 là vị trí bắt đầu xóa.
  - Tham số 2 là số lượng phần tử cần xóa.
  - Tham số 3 là phần tử muốn thêm vào vị trí phần tử đã xóa.
- arr.splice() trả về mảng các phần tử bị xóa.
- Ví dụ:

```
function htmlColorNames(arr) {
        arr.splice(0, 2,'DarkSalmon', 'BlanchedAlmond');
        return arr;
}
console.log(htmlColorNames(['DarkGoldenRod', 'WhiteSmoke', 'LavenderBlush', 'PaleTurqoise',
'FireBrick']));
```

Câu 5.1.3. [slice()]

- slice(), rather than modifying an array, copies, or extracts, a given number of elements to a new array, leaving the array it is called upon untouched.
- slice() takes 2 parameters
  - The first is the index at which to begin extraction.

- The second is the index at which to stop extraction (extraction will occur up to, but not including the element at this index).

- Example:
  let weatherConditions = ['rain', 'snow', 'sleet', 'hail', 'clear'];
  let todaysWeather = weatherConditions.slice(1, 3);
  // todaysWeather equals ['snow', 'sleet'];
  // weatherConditions still equals ['rain', 'snow', 'sleet', 'hail', 'clear']

Câu 5.1.4. […]

- We can use the spread operator to copy an array like so:
  let thisArray = [true, true, undefined, false, null];
  let thatArray = [...thisArray];
  // thatArray equals [true, true, undefined, false, null]
  // thisArray remains unchanged, and is identical to thatArray
- Or we can use it to combine arrays, or to insert all the elements of one array into another, at any index.
  let thisArray = ['sage', 'rosemary', 'parsley', 'thyme'];
  let thatArray = ['basil', 'cilantro', ...thisArray, 'coriander'];
  // thatArray now equals ['basil', 'cilantro', 'sage', 'rosemary', 'parsley', 'thyme', 'coriander']
- With more traditional syntaxes, we can concatenate arrays, but this only allows us to combine arrays at the end of one, and at the start of another.

Câu 5.1.5. [indexOf()]

- indexOf() checks for the presence of an element on an array. indexOf() takes an element as a parameter, and when called, it returns the position, or index, of that element, or -1 if the element does not exist on the array.

5.2. Đối tượng

- Objects are collections of key-value pairs.
- We can access a property by using dot notation (A.B). Alternatively, we can also access a property with brackets (A['B']).
- With bracket notation, we enclosed followers in quotes because the brackets allow us to pass a variable in to be evaluated as a property name.

Câu 5.2.1. [Nested object]

- Object properties can be arrays or other objects.
- For example:

```
let A = {
        A1: 1,
        A2: {
                A21: 1,
                A22: 2
        }
}
```

Câu 5.2.2. [[]]

- Bracket notation can access property values using the evaluation of a variable.
- For example:

```
let A = {
        A1: 1,
        A2: 2
}

function F(X){
        A[X] = 2;
}

F('A1');
```

Câu 5.2.3. [delete]

- delete A.A1;

Câu 5.2.4. [hasOwnProperty - in]

- There are two different ways to check if an object has a specific property.

- One uses the hasOwnProperty() method and the other uses the in keyword.
  A.hasOwnProperty('A1');
  'A1' in A;

Câu 5.2.5. [for...in]

- We can iterate through all the keys within an object by a for...in statement.
- For example:
  for(let X in A){
  doSomething(A[X]);
  }
- Note: In the exercise, we use obj[user].online instead of just user.online.

Câu 5.2.6. [Object.keys()]

- We can generate an array which contains all the keys stored in an object using the Object.keys().
- This will return an array with strings representing each property in the object. There will be no specific order to the entries in the array.
- For example:
  let B = Object.keys(A);

TỔNG KẾT

❖ Sử dụng các hàm sau để thao tác trên MẢNG.

- **arr.push():** Thêm phần tử vào cuối mảng.
  VD: arr.push(1);
- **arr.unshift():** Thêm phần tử vào đầu mảng.
  VD: arr.unshift(1);
- **arr.pop():** Xóa phần tử cuối mảng, return phần tử này.
  VD: let x = arr.pop();
- **arr.shift():** Xóa phần tử đầu mảng, return phần tử này.
  VD: let x = arr.shift();
- **arr.splice():** Xóa dãy các phần tử liên tiếp bất kỳ. Hàm này nhận 3 tham số là (vitribatdauxoa, soluongcanxoa, daycacphantuthaythe).
  VD: arr.splice(1, 2, 3, 4);
- **arr.slice():** Trích xuất dãy các phần từ liên tiếp từ mảng. Hàm này nhận 2 tham số là (vitribatdau, vitriketthucnhungkhonglayphantunay).
  VD: let newArr = arr.slice(1, 2);
- **[...]** có thể được dùng để copy một mảng.
  VD: let newArr = [...arr];
- **arr.indexOf():** Tìm chỉ số của một phần tử. Trả về -1 nếu phần tử này không tồn tại.
  VD: let index = arr.indexOf(x);

❖ Các thao tác sau có thể được thực hiện trên ĐỐI TƯỢNG.

- Sử dụng **A.A1** hoặc **A['A1']** để truy xuất đến các phần tử của đối tượng. Sử dụng . khi biết tên thuộc tính. Sử dụng [] khi thuộc tính là tham số/ ẩn số.
- Sử dụng **[delete]** để xóa thuộc tính của đối tượng.
- Sử dụng **obj.hasOwnProperty()** để kiểm tra đối tượng có chứa thuộc tính hay không.
  VD: A.hasOwnProperty('A1');
- Sử dụng vòng lặp **for...in** để duyệt qua các thuộc tính của một đối tượng.
  VD: for(let X in A){doSomething(A[X]);}
- Sử dụng **Object.keys()** để lấy mảng các thuộc tính (key) của một đối tượng.
  VD: let arr = Object.keys(A);

6. Basic Algorithm Scripting

Câu 6.1. {C2F - chuyển độ C thành độ F}

- F = 9/5*C + 32
- Đáp án:

```
function convertToF(celsius) {
      var fahrenheit = (celsius * (9/5)) + 32;
      if ( typeof fahrenheit !== 'undefined' ) {
            return fahrenheit;
      } else {
```

```
            return 'fahrenheit not defined';
        }
convertToF(30);
```

Câu 6.2. {Reverse a string - Đảo ngược một xâu}

- split() được dùng để chuyển xâu thành mảng. Ví dụ:
  'ABC'.split('') ==== ['A', 'B', 'C'];
  'A B C'.split(' ') ==== ['A', 'B', 'C'];
- Xâu có một số tính chất giống mảng, chẳng hạn, có thể truy xuất đến các phần tử theo cú pháp A[X] hay tính độ dài theo cú pháp A.length. Tuy nhiên, xâu không có một số hàm đặc trưng của mảng, trong đó có reverse().
- join() được dùng để chuyển mảng thành xâu. Ví dụ:
  let A = ['A', 'B', 'C'];
  A.join('') ==== 'ABC'
- Vậy, để đảo một xâu. Ta dùng cú pháp: A.split('').reverse().join('');
- Đáp án:

```
function reverseString(str) {
  return str.split('').reverse().join('');
}
```

Câu 6.3. {Factorialize a Number - Giai thừa}

- n! = n*(n-1)*...*1;
- Đáp án:

```
function factorialize(num) {
  if (num === 0) { return 1; }
  return num * factorialize(num-1);
}

factorialize(5);
```

Câu 6.4. {Longest word in a string - Từ dài nhất trong một xâu}

- Chuyển xâu thành mảng bằng hàm split(' ') để thu được các từ.
- Duyệt qua các phần tử.
- Đáp án:

```
function findLongestWordLength(str) {
  var words = str.split(' ');
  var maxLength = 0;

  for (var i = 0; i < words.length; i++) {
    if (words[i].length > maxLength) {
      maxLength = words[i].length;
    }
  }

  return maxLength;
}
```

HOẶC

```
function findLongestWordLength(s) {
  return s.split(' ')
    .reduce(function(x, y) {
      return Math.max(x, y.length)
    }, 0);
}
```

Câu 6.5. {Return Largest Numbers in Arrays - Phần tử lớn nhất trong các mảng}

- **Đề bài:** Cho một mảng 2 chiều. Trả về mảng chứa phần tử lớn nhất trong mỗi mảng con.
- Duyệt qua mỗi mảng con.
- Tìm phần tử lớn nhất trong từng mảng này.

- Thêm phần tử lớn nhất vào mảng kết quả.
- Đáp án:

```
function largestOfFour(arr) {
  var results = [];
  for (var n = 0; n < arr.length; n++) {
    var largestNumber = arr[n][0];
    for (var sb = 1; sb < arr[n].length; sb++) {
      if (arr[n][sb] > largestNumber) {
        largestNumber = arr[n][sb];
      }
    }

    results[n] = largestNumber;
  }

  return results;
}
```

HOẶC

```
function largestOfFour(arr) {
  return arr.map(function(group){
    return group.reduce(function(prev, current) {
      return (current > prev) ? current : prev;
    });
  });
}
```

Câu 6.5. {Confirm the Ending - Xác nhận kết thúc xâu}

- **Đề bài:** Kiểm tra xem một xâu có được kết thúc bằng một xâu nào đó hay không.
- Tính độ dài của 2 xâu.
- Kiểm tra xem xâu con từ vị trí (len1 - len2) đến cuối xâu 1 có trùng với xâu 2 hay không.
  Cụ thể: str1.substring(str1.length - str2.length) === str2?
- Lưu ý: Hàm A.substring(x, y) sẽ cắt xâu từ vị trí x, x+1,..., y-1. Nếu không có tham số thứ 2 thì nó sẽ cắt từ x đến cuối.
- Đáp án:

```
function confirmEnding(str, target) {
  return str.slice(str.length - target.length) === target;
}

confirmEnding("He has to give me a new name", "name");
```

Câu 6.6. {Repeat a String Repeat a String - Lặp một xâu}

- Khởi tạo xâu kết quả bằng rỗng.
- Sử dụng vòng lặp cộng vào xâu kết quả n lần.
- Đáp án:

```
function repeatStringNumTimes(str, num) {
  var accumulatedStr = '';

  while (num > 0) {
    accumulatedStr += str;
    num--;
  }

  return accumulatedStr;
}
```

HOẶC

```
function repeatStringNumTimes(str, num) {
  if(num < 0)
    return "";
  if(num === 1)
    return str;
  else
    return str + repeatStringNumTimes(str, num - 1);
}
```

Câu 6.7. {Truncate a String - Cắt một xâu}

- **Đề bài:** Cắt một xâu nếu nó vượt quá một độ dài tối đa nào đó và thêm ... vào cuối xâu vừa cắt.
- Sử dụng substring() và +
  Cụ thể: return num < str.length? str.substring(0, num) + "...": str;
- Đáp án:

```
function truncateString(str, num) {
  // Clear out that junk in your trunk
  if (str.length > num && num > 3) {
    return str.slice(0, (num - 3)) + '...';
  } else if (str.length > num && num <= 3) {
    return str.slice(0, num) + '...';
  } else {
    return str;
  }
}
```

HOẶC

```
function truncateString(str, num) {
  if (str.length <= num) {
    return str;
  } else {
    return str.slice(0, num > 3 ? num - 3 : num) + '...';
  }
}
```

Câu 6.8. {Finders Keepers}

- **Đề bài:** Tạo hàm duyệt qua một mảng và trả về phần tử đầu tiên trong mảng thỏa mãn một biểu thức nào đấy. Trả về underfined nếu không có phần tử nào thỏa mãn.
- Ví dụ: findElement([1, 2, 3, 4], num => num % 2 === 0);
  function findElement(arr, func) {}
- Duyệt qua các phần tử của mảng. Trả về phần tử đầu tiên thỏa mãn biểu thức. Lưu ý sử dụng break vì các phần tử sau vẫn có thể thỏa mãn điều kiện.
- Lưu ý: Ở đây, hàm đóng vai trò tham số.
- Đáp án:

```
function findElement(arr, func) {
  let num = 0;

  for(var i = 0; i < arr.length; i++) {
    num = arr[i];
    if (func(num)) {
      return num;
    }
  }

  return undefined;
}
```

Câu 6.9. {Boo who}

- **Đề bài:** Kiểm tra xem một giá trị nào đấy có thuộc kiểu boolean hay không.
- Sử dụng từ khóa typeof. Cụ thể: typeof bool == 'boolean';
- Đáp án:

```
function booWho(bool) {
   return typeof bool === 'boolean';
}
// test here
booWho(null);
```

Câu 6.10. {Title Case a Sentence}

- **Đề bài:** Cho một xâu bất kỳ. Trả về xâu tương ứng có chữ cái đầu tiên của mỗi từ được viết hoa và các từ còn lại được viết thường.
- Đầu tiên, chuyển xâu về thành mảng các tử.
- Duyệt qua các phần tử của mảng. Dùng hàm toUpperCase() để chuyển ký tự đầu tiên trong mảng về chữ hoa. Dùng hàm toLowerCase() để chuyển xâu con từ vị trí thứ 2 đến cuối về chữ thường. Nối 2 xâu này lại vào thêm vào mảng mới.
- Sử dụng join(' ') để thu được xâu từ mảng này.
- Lưu ý: Không dùng + để nối xâu vì sẽ thiếu các dấu ' '
- Đáp án:

```
function titleCase(str) {
  return str.toLowerCase().replace(/(^|\s)\S/g, (L) => L.toUpperCase());
}
```

Câu 6.11. {Slice and Splice}

- **Đề bài:** Thêm các phần tử của mảng này vào mảng khác tại một chỉ số nào đó sử dụng slice() và splice().
- Đầu tiên, slice() toàn bộ các phần tử của mảng 2 vào một mảng cục bộ.
- Tổ chức một vòng lặp để thêm lần lượt từng phần tử của mảng 1 vào mảng 2 ở vị trí n (n tăng dần sau mỗi vòng lặp).
- Lưu ý: B = A.slice() cho phép copy toàn bộ dữ liệu mảng A vào mảng B. Mọi thao tác trên B sẽ không ảnh hưởng đến A và ngược lại. Nếu sử dụng cú pháp B = A thì mọi thao tác trên B sẽ ảnh hưởng đến A.
- Đáp án:

```
function frankenSplice(arr1, arr2, n) {
  // It's alive. It's alive!
  let localArray = arr2.slice();
  for (let i = 0; i < arr1.length; i++) {
    localArray.splice(n, 0, arr1[i]);
    n++;
  }
  return localArray;
}
```

Câu 6.12. {Falsy Bouncer}

- **Đề bài:** Xóa các giá trị false, null, 0, "", undefined và NaN từ một mảng.
- Duyệt qua từng phần tử trong mảng.
- Sử dụng toán tử !! để chuyển 1 giá trị bất kỳ về boolean.
- Xóa phần tử bằng splice(vị trí phần tử, 1).
- Lưu ý: Khi xóa xong phần tử, cần giảm chỉ số xuống 1 để sau vòng lặp, phần tử được xét là phần tử vừa được dịch lên.
- Đáp án:

```
function bouncer(arr) {
  return arr.filter(Boolean);
}
```

Câu 6.13. {Where do I Belong}

- **Đề bài:** Trả về vị trí chỉ số thấp nhất mà tại đây, một giá trị nào đó sẽ được chèn vào đó nếu mảng được sắp xếp. Chẳng hạn:
  getIndexToIns([1,2,3,4], 1.5) sẽ trả về 1 do 1.5 sẽ ở vị trí thứ 2 (chỉ số 1) nếu nó được thêm vào mảng và mảng được sắp xếp lại.
  getIndexToIns([20,3,5], 19) sẽ trả về 2 (3, 5, 19, 20).
- PHÁT HIỆN VĨ ĐẠI: Vị trí của số này chính là số lượng phần tử nhỏ hơn nó.
- Lưu ý: Không cần phải sắp xếp lại mảng.
- Đáp án:

```
function getIndexToIns(arr, num) {
  arr.sort(function(a, b) {
```

```
    return a - b;
  });

  for (var a = 0; a < arr.length; a++) {
    if (arr[a] >= num)
      return a;
  }

  return arr.length;
}
```

HOẶC

```
function getIndexToIns(arr, num) {
  arr.push(num);
  arr.sort(function(a, b){return a-b});
  return arr.indexOf(num);
}
```

HOẶC

```
function getIndexToIns(arr, num) {
   return arr.concat(num).sort((a,b) => a-b).indexOf(num);
}

getIndexToIns([1,3,4],2);
```

Câu 6.14. {Mutations}

- **Đề bài:** Cho một mảng [A, B] trong đó A, B là các xâu. Trả về true nếu A chứa mọi ký tự trong B.
- Cách 1: Chuyển xâu về mảng các ký tự. Tổ chức vòng lặp để so khớp từng ký tự trong xâu 2 với các ký tự trong xâu 1. Lưu ý: So sánh cần sử dụng toLowerCase().
- Cách 2: Sử dụng indexOf()
- Đáp án:

```
function mutation(arr) {
  var test = arr[1].toLowerCase();
  var target = arr[0].toLowerCase();
  for (var i=0;i<test.length;i++) {
    if (target.indexOf(test[i]) < 0)
      return false;
  }
  return true;
 }
```

HOẶC

```
function mutation(arr) {
  return arr[1].toLowerCase()
    .split('')
    .every(function(letter) {
      return arr[0].toLowerCase()
        .indexOf(letter) != -1;
    });
}
```

Câu 6.15. {Chunky Monkey}

- **Đề bài:** Chia mảng thành các mảng con có kích thước n và lưu chúng vào một mảng mới.
- Tổ chức vòng lặp for, môi vòng lặp tăng biến đếm (i) lên một giá trị size. Cắt mảng con từ i đến i + size và push vào mảng mới.

- Cụ thể:

```
let newArr = [];
for(let i = 0; i < arr.length; i += size)
        newArr.push(arr.slice(i, i + size));
```

- Đáp án:

```
function chunkArrayInGroups(arr, size) {
  // Break it up.
  var arr2 = [];
  for (var i = 0; i < arr.length; i+=size) {
    arr2.push(arr.slice(i , i+size));
  }
  return arr2;
}
```

7. Object Oriented Programming

Câu 7.1. [object]

❖ **[object]**

```
let A = {
        A1: x,
        A2: y
}
```

❖ **[.]**

```
A.A1
```

❖ **[method]**

- Objects can have a special type of property, called a method.
- Methods are properties that are functions. This adds different behavior to an object.

```
let A = {
        A1: x,
        A2: y,
        A3: function() {doSomething;}
}
```

❖ **[this]**

```
let A = {
        A1: x,
        A2: y,
        A3: function() {return "A1 = " + this.A1;}
};
```

Câu 7.2. [constructor]

❖ **[constructor]**

- Constructors are functions that create new objects.

```
function A() {
        this.A1 = x;
        this.A2 = y;
}
```

- Constructors follow a few conventions:
  - Constructors are defined with a capitalized name to distinguish them from other functions that are not constructors.
  - Constructors use the keyword this to set properties of the object they will create. Inside the constructor, this refers to the new object it will create.
  - Constructors define properties and behaviors instead of returning a value as other functions might.
- Constructors accept parameters:
  - function A(X, Y) {
    this.A1 = X;
    this.A2 = Y;
    }

- We can pass in the values as arguments to define each unique object:

```
let newA = new A(x, y);
```

❖ **[create object]**

- We can create an object using a constructor:
  let newA = new A();
- newA has all the properties defined inside the A constructor:
  newA.A1;
- Just like any other object, its properties can be accessed and modified: newA.A1 = z;

❖ **[instanceof]**

- We can use instanceof to compare an object to a constructor. It returns true or false based on whether or not that object was created with the constructor.

```
newA instanceof A; //true
```

- If an object is created without using a constructor, instanceof will verify that it is not an instance of that constructor:

```
let anotherA = {
        A1: x,
        A2: y
}
anotherA instanceof A; //false
```

Câu 7.3. [hasOwnProperty]

```
let B = [];
for(let prop in newA) {
        if(newA.hasOwnProperty(prop)) {
                B.push(prop);
        }
}
// B = ["A1", "A2"]
```

Câu 7.3. [prototype]

- Some properties have the same value for all instances of an object, we can use prototype to reduce the number of duplicated variables.
- The prototype is ***an object that is shared among ALL instances of an object.*** For example:

```
function A() {
        this.A1 = x;
        this.A2 = y;
}
A.prototype.A3 = 2;
```

  Now all instances of A have the A3 property.
- Nearly every object in JavaScript has a prototype property which is part of the constructor function that created it.

Câu 7.4. [own/prototype]

- There are two kinds of properties: ***own properties and prototype properties.***
  - Own properties are defined directly on the object instance itself.
  - And prototype properties are defined on the prototype.
- We can iterate over all properties:

```
let ownB = [];
let prototypeB = []
for(let prop in newA) {
        if(newA.hasOwnProperty(prop)) {
                ownB.push(prop);
        } else {
                prototypeB.push(prop);
        }
```

```
}
// ownB = ["A1", "A2"]; prototypeB = ["A3"]
```

- We can set the prototype to a new object that already contains the properties.

```
A.prototype = {
    A3: z,
    A4: function() {
        doSomething;
    },
    A5: function() {
        doSomething;
    }
};
```

- **{Note}**
  - When setting the prototype to a new object, it erases the constructor property.

```
console.log(A.constructor) //undefined
```

  - To fix this, whenever a prototype is set to a new object, remember to define the constructor property:

```
A.prototype = {
    constructor: A,
    A3: z,
    ....
};
```

Câu 7.6. [constructor property]

- Constructor property is a reference to the constructor function that created the instance.
- It is used to ***find out what kind of object the instance is***. (like instanceof)

```
if(newA.constructor === A) {
    return true;
} else {
    return false;
}
```

Câu 7.7. [isPrototypeOf]

- An object inherits its prototype directly from the constructor function that created it.

```
A.prototype.isPrototypeOf(newA); //true
```

Câu 7.8. [prototype chain]

- Because a prototype is an object, a prototype can have its own prototype. In this case, the prototype of A.prototype is Object.prototype:

```
Object.prototype.isPrototypeOf(A.prototype); //true
```

- The hasOwnProperty method is defined in Object.prototype, which can be accessed by A.prototype, which can then be accessed by newA. This is an example of the prototype chain.
- In this prototype chain, A is the supertype for newA, while newA is the subtype. Object is a supertype for both A and newA.
- Object is a supertype for all objects in JavaScript. Therefore, any object can use the hasOwnProperty method.
- Minh họa:
  Object = {..., prototype: {...}}
  A = {..., prototype: {..., protype: {}}}
  Prototype của A.prototype (cũng là một đối tượng) là Object.prototype.

Câu 7.9. [supertype]

- We can create a supertype (or parent) to avoid repeated code.
- For example:

```
function A(X) {this.A1 = X;}
A.prototype = {constructor A, A2: function() {doSomething;}};

function anotherA(X) {this.A1 = X;}
anotherA.prototype = {constructor anotherA, A2: function() {doSomething;}};
```

- ... can change to:

```
function superA() {}
superA.prototype = {
        constructor superA,
        A2: function() {doSomething;}
}
```

Câu 7.10. [inheritance]

- We can reuse superA's methods inside A and anotherA without defining them again.
- **Step 1:** Make an instance of the supertype (or parent)

```
let newA = Object.create(superA.prototype);
```

  - Object.create(obj) creates a new object, and sets obj as the new object's prototype.
  - By setting the prototype of newA to be superA's prototype, you are giving the newA instance the same "recipe" as any other instance of superA.

```
newA.A2();
newA instanceof superA; // => true
```

- **Step 2:** Set the prototype of the subtype (or child) to be an instance of supertype.

```
A.prototype = Object.create(superA.prototype);
```

- When an object inherits its prototype from another object, it also inherits the supertype's constructor property.

```
A.constructor; // function superA(){...}
```

  - We should edit A's constructor property to the A object

```
A.prototype.constructor = A;
```

Câu 7.11. [add methods after inheritance]

- A constructor function that inherits its prototype object from a supertype constructor function can still have its own methods in addition to inherited methods.
- ***For example:***

```
function superA() {}
superA.prototype.F = function(){doSomething;};
function A() {}
A.prototype = Object.create(superA.prototype);
A.prototype.constructor = A;
```

- We can add behavior that is unique to A objects.

```
A.prototype.otherF = function(){doSomething;};
```

- Now: let newA = new A() will have both F() and otherF() methods.

Câu 7.12. [override]

- We can override an inherited method by adding it to ChildObject. prototype using the same method name as the one to override.

Câu 7.13. [mixin]

- Inheritance does not work well for unrelated objects.
- For them, it's better to use mixins. A mixin allows other objects to use a collection of functions.

```
let fMixin = function(obj) {
    obj.F = function() {doSomething;}
}
fMixIn(A);
fMixIn(B);
```

Câu 7.14. [protect properties/closure]

- To make properties private, we can create them within the constructor function.

```
function A() {
        let A1 = x;
        this.getA1() {
                return A1;
        }
}
let newA = new A();
newA.getA1();
```

- In JavaScript, a function always has access to the context in which it was created. This is called closure.

Câu 7.15. [IIFE/Immediately invoked function expression]

- We can execute a function as soon as it is declared:

  (function() {doSomething();})();
- The two parentheses () at the end cause it to be immediately executed or invoked.
- IIFE is often used to group related functionality into a single object or module. For example, we can group mixins into a module:

```
let model = (function() {
        return {
                F1Mixin: function(obj) {obj.F1 = function() {doSomething();};},
                F2Mixin: function(obj) {obj.F2 = function() {doSomething2();};}
        }
})();
model.F1Mixin(newA);
newA.F1();
```

8. Functional Programming

Câu 8.1. [functional programming]

- Functional programming is a style of programming where solutions are simple, isolated functions, without any side effects outside of the function scope.

  INPUT -> PROCESS -> OUTPUT

  Functional programming is about:

  1) Isolated functions - there is no dependence on the state of the program, which includes global variables that are subject to change

  2) Pure functions - the same input always gives the same output

  3) Functions with limited side effects - any changes, or mutations, to the state of the program outside the function are carefully controlled
- ***[imperating code]***
  - Imperative programming gives the computer a set of statements to perform a task. Often the statements change the state of the program, like updating global variables.
  - In contrast, functional programming is a form of declarative programming. You tell the computer what you want done by calling a method or function.
- ***[avoid mutations and side effects]***
  - One of the core principle of functional programming is to **not change things.** Changes lead to bugs. Don't change anything, including the function arguments or any global variable.
  - In functional programming, changing or altering things is called mutation, and the outcome is called a side effect.
  - A function, ideally, should be a pure function, meaning that it does not cause any side effects.
- ***[pass arguments to avoid external dependence]***
  - Another principle of functional programming is to always declare your dependencies explicitly. If a function depends on a variable or object being present, then **pass that variable or object directly into the function as an argument.**
- ***{Lưu ý}***

  1) Don't alter a variable or object - create new variables and objects and return them if need be from a function.

  2) Declare function arguments - any computation inside a function depends only on the arguments, and not on any global object or variable.
- ***{Lưu ý}***
  - Khi truyền một mảng vào một hàm, để không làm thay đổi giá trị của mảng gốc, ta nên khai báo một biến bảng mới và thao tác trên biến này. Chẳng hạn:

```
function f(arr) {
        let newArr = [...arr];
}
```

  - Nếu khai báo: let f = (x) => x; thì f chính là tên hàm. Hàm này có thể viết thành function f(x) {return x;}

- Nếu muốn gọi đến hàm F = (f, y) => f(y)*y; ta truyền f vào tham số thay vì f().
  Ví dụ: F(f,1);

Câu 8.2. [map()]

- The map method iterates over each item in an array. It creates a new array (without changing the original one) after applying a callback function to every element.
- The map method takes a function as in input and returns an array. The returned array includes elements that is processed by the function. This function takes individual elements as input.
- For example:

```
let arr = [1, 2, 3]
let newArr = arr.map((item) => item + 1);                    // newArr = [2, 3, 4]
```

- map is a pure function. Its output depends solely on its inputs. Plus, it takes another function as its argument.
- **Note:** A pure function is allowed to alter local variables defined within its scope, although, it's preferable to avoid that as well.
- **{Lưu ý}** Ta có thể duyệt qua từng phần tử của một mảng thông qua hàm Array.prototype.forEach() hay forEach().

```
var arr = [1, 2];
arr.forEach(function(element) {
        console.log(element);
});
```

- **{Lưu ý}** Khi viết hàm được khai báo kiểu Array.prototype.function, để trỏ đến đối tượng, ta sử dụng this.

Câu 8.3. [filter()]

- Array.prototype.filter(), or filter() takes a callback function that applies the logic inside the callback on each element of the array as an argument.
- If an element returns true based on the criteria in the callback function, then it is included in the new array.
- For example:

```
let arr = [1, 2, 3]
let newArr = arr.filter((item) => (item > 2) && (item < 4));
// newArr = [3]
```

- Có thể kết hợp filter() và map() cùng nhau khi cần thiết. Chẳng hạn:

```
let arr = [1, 2, 3, 4];
arr.filter((item) => (item%2 === 0));
arr.map((item) => (item*item));
```

Câu 8.4. [slice()]

- slice method returns a copy of certain elements of an array.
- It can take two arguments, the index of where to begin the slice and the index for where to end the slice (non-inclusive).
- If the arguments are not provided, the default is to start at the beginning of the array through the end, which is an easy way to make a copy of the entire array. The slice method does not mutate the original array, but returns a new one.
- ***[splice()]***
  - To remove items from array, we use splice() method, which takes arguments for the index of where to start removing items, then the number of items to remove. (If the second argument is not provided, the default is to remove items through the end.)
  - However, the splice method mutates the original array it is called on.
  - Meanwhile, the slice method does not mutate the original array, but returns a new one which can be saved into a variable.
  - We can use slice method instead of splice to avoid any array-mutating side effects.

Câu 8.5. [concat()]

- We can use concat() method to join two strings, or two arrays.
- For arrays, the method is called on one, then another array is provided as the argument to concat, which is added to the end of the first array.
- It returns a new array and does not mutate either of the original arrays. For example: [A1, A2].concat([B1, B2]);
- Compare concat() to the push() method.
  - Push adds an item to the end of the same array it is called on, which mutates that array.

- Concat offers a way to add new items to the end of an array without any mutating side effects.

Câu 8.6. [reduce()]

- We can solve almost any array processing problem using the reduce method because unlike filter and map, reduce allow interaction between two different elements of the array.
- For example, if you want to compare elements of the array, or add them together, filter or map could not process that.
- In other words, both filter and map are special applications of reduce.
- Ví dụ:
  let A = [{P1: x1, P2: x2}, {P1: y1, P2: y2}, {P1: x1, P2: z2}, {P1: y1, P2: z2}];
  Để tính tổng các giá trị P2 của các đối tượng có P1 = x1, ta làm như sau:
  A.filter(x => x.P1 === x1).map(x => x.P2).reduce((x1, x2) => x1 + x2);

Câu 8.7. [sort()]

- The sort method sorts the elements of an array according to the callback function.
- It's encouraged to provide a callback function to specify how to sort the array items. JavaScript's default sorting method is by string Unicode point value, which may return unexpected results.
- Ví dụ:
  - arr.sort(); //Sap xep theo chieu tang dan
  - arr.sort((x, y) => x - y); //Sap xep theo chieu tang dan
  - arr.sort((x, y) => x < y); //Sap xep theo chieu giam dan
- Sử dụng sort có thể làm thay đổi thứ tự các phần tử ở mảng gốc. Do đó, trước khi sort, ta nên sử dụng concat để nối mảng rỗng vào mảng cần sắp xếp (do concat trả về mảng mới).

  A.concat([]);
  A.sort();

Câu 8.8. [split()]

- The split method splits a string into an array of strings. It takes an argument for the delimiter, which can be a character to use to break up the string or a regular expression.
- For example, if the delimiter is a space, you get an array of words, and if the delimiter is an empty string, you get an array of each character in the string.

  arr.split(" ");
  arr.split("");
  arr.split(/\d/);
- Để tách các từ trong một câu (có thể được ngăn cách bởi các khoảng trắng hoặc các dấu câu), ta có thể sử dụng lệnh:

  arr.split(/\W/)

Câu 8.9. [join()]

- The join method is used to join the elements of an array together to create a string.
- It takes an argument for the delimiter that is used to separate the array elements in the string.
- For example:

  let arr = ["Hi", "bitch"];
  arr.join(" ");
  //Return "Hi bitch"

Câu 8.9. {Bài tập – URL}

- Many content management sites (CMS) have the titles of a post added to part of the URL for simple bookmarking purposes.
- For example, if you write a Medium post titled "Stop Using Reduce", it's likely the URL would have some form of the title string in it ("".../stop-using-reduce").
- **{Đáp án}**

  title.toLowerCase().trim().split(/\s+/).join("-");

Câu 8.10 [every()]

- The every method works with arrays to check if every element passes a particular test.
- It returns a Boolean value - true if all values meet the criteria, false if not.
- For example:

  let A = [1, 2];
  A.every(x => x > 2); // return false

Câu 8.11. [some()]

- The some method works with arrays to check if any element passes a particular test.
- It returns a Boolean value - true if any of the values meet the criteria, false if not.
- For example:

```
let A = [1, 2];
A.some(x => x > 1);                    // return true
```

Câu 8.12. [currying]

- The arity of a function is the number of arguments it requires.
- Currying a function means to convert a function of N arity into N functions of arity 1. In other words, it restructures a function so it takes one argument, then returns another function that takes the next argument, and so on.
- For example:

```
function unCurried(x, y) {return x + y;}
function curried(x) { return function(y) {return x + y;}}
curried(1)(2)
```

- To supply all the arguments to a function at one time, you can save each function call into a variable, which will hold the returned function reference that takes the next argument when it's available.
- For example:

```
var funcForY = curried(1);
console.log(funcForY(2));
```

- Ví dụ, để cộng 3 số, ta có thể viết:

```
function add(x) {return y => z => z + y + x;};
```

TỔNG KẾT

❖ **[ĐỐI VỚI MẢNG]**

- Sử dụng arr.map() để duyệt qua các phần tử của một mảng và nhận về mảng kết quả sau khi áp dụng hàm call back lên các phần tử của mảng này.
  VD: let newArr = arr.map(x => x + 1);
- Sử dụng arr.filter() để lọc các phần tử thỏa mãn một điều kiện nào đó (được cho bởi hàm callback) từ một mảng cho trước.
  VD: let newArr = arr.filter(x => x > 0);
- Sử dụng arr.slice() để trích xuất một dãy các phần tử từ một mảng cho trước. Để tránh làm thay đổi mảng gốc, ta có thể dùng slice() trước khi thực hiện splice().
  VD: let newArr = arr.slice();
- Sử dụng arr.concat() để nối hai mảng.
  VD: let newArr = arr1.concat(arr2);
- Sử dụng arr.reduce() khi muốn thao tác với nhiều hơn 1 phần tử trên một mảng.
  VD: let x = arr.reduce((x, y) => x + y);
- Sử dụng arr.sort() để sắp xếp các phần tử của mảng. sort() không có tham số sẽ trả về giá trị tăng dần.
  VD: arr.sort((x, y) => x < y); // Sap xep theo chieu giam dan
- Sử dụng arr.every() để kiểm tra xem MỌI phần tử của một mảng có thỏa mãn ràng buộc (được cho bởi hàm callback) hay không.
  VD: arr.every(x => x > 0);
- Sủ dụng arr.some() để kiểm tra xem mảng có ÍT NHẤT một phần tử thỏa mãn ràng buộc (được cho bởi hàm callback) hay không.
  VD: arr.some(x => x > 0)

❖ **[ĐỐI VỚI XÂU]**

- Sử dụng str.split() để phân chia xâu thành mảng các xâu con. Ký hiệu phân tách có thể là một ký tự, một xâu hay một regex.
  - Sử dụng let chars = str.split("") để tách câu thành mảng các các ký tự.
  - Sử dụng let words = str.split(" ") để tách câu thành mảng các từ.
  - Sử dụng let words = str.split(/\W/) để tách câu thành mảng các từ bị ngăn cách bởi ký hiệu phi ký tự.
- Sử dụng str.join() để nối các phần tử trong một mảng về thành một xâu.
  VD: let str = arr.join(" ");

❖ **[LƯU Ý]**

- splice() chỉ dùng cho mảng.
- toLowerCase() và toUpperCase() KHÔNG làm thay đổi xâu gốc.

9. <u>Intermediate Algorithm Scripting</u>

Câu 9.1. {Sum All Numbers in a Range - Tổng trong một khoảng}

- **Đề bài:** Cho một mảng gồm 2 số. Trả về tổng của chúng và các số nằm giữa chúng. Lưu ý, số nhỏ hơn không hẳn đã ở trước.
- Ý tưởng:
  - Tìm số nhỏ hơn (và số lớn hơn).
  - Tổ chức vòng lặp để tính tổng.

Câu 9.2. {Diff Two Arrays - Hiệu hai mảng}

- **Đề bài:** So sánh hai mảng và trả về mảng các phần tử chỉ xuất hiện ở mảng này nhưng không xuất hiện ở mảng kia.
- Ý tưởng:
  - Ý tưởng 1. concat 2 mảng. Sử dụng filter để lọc.
  - Ý tưởng 2.
    filter các phần tử thuộc mảng 1 không thuộc mảng 2.
    filter các phần tử thuộc mảng 2 không thuộc mảng 1.
    concat chúng.
- **{Lưu ý}** arr.includes(x) được dùng để kiểm tra xem x có thuộc arr hay không.
- **Đáp án:**

```
function diffArray(arr1, arr2) {
  return arr1
       .concat(arr2)
       .filter(
             item => !arr1.includes(item) || !arr2.includes(item)
       )
}
```

Câu 9.3. {Seek and Destroy - Tìm và xóa}

- **Đề bài:** Cho một mảng và một số tham số. Tìm và xóa các phần tử có cùng giá trị với các tham số này.
- Hàm chỉ có dạng: function A(arr), cần sử dụng đối tượng arguments để tham chiếu đến các tham số.
- Ý tưởng:
  - Đưa các tham số vào một mảng. (Lưu ý, xóa tham số đầu)
    Sử dụng filter và include để lọc các phần tử.
  - Sử dụng filter để lọc các phần tử.
    Tổ chức vòng lặp để duyệt qua các tham số.
- **{Lưu ý}**
  - let args = Array.from(arguments).slice(1);
    arr.filter(x => !args.includes(x));
  - arr.from(đối tượng) trả về mảng.
- **Đáp án:**

```
function destroyer(arr) {
  var args = Array.prototype.slice.call(arguments);
  for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < args.length; j++) {
      if (arr[i] === args[j]) delete arr[i];
    }
  }
  return arr.filter(Boolean);
}
```

HOẶC

```
const destroyer = (arr, ...args) => arr.filter(i => !args.includes(i));
```

Câu 9.4. {Wherefore art thou}

- **Đề bài:** Tạo hàm duyệt qua một mảng các đối tượng (tham số đầu) và trả về mảng các đối tượng có cùng các cặp tên và giá trị (tham số hai).
  Ví dụ: Tham số 1 là [{A: x, B: y}, {B: y}, {B: x, C: z}], tham số 2 là {A: x, B: y}. Hàm sẽ trả về [{A: x, B: y}]
- Ý tưởng
  - Lưu lại danh sách các key ở tham số 2.
  - Tổ chức vòng lặp:
    Lọc ra các đối tượng có key 1 = value 1
    Lọc ra các đối tượng có key 2 = value 2
  - return collection.filter(function(obj) {
    return keys.every(function(key) {
    return obj.hasOwnProperty(key) && obj[key] == source[key];
    });
    });
- **Đáp án:**

```
function whatIsInAName(collection, source) {
  var srcKeys = Object.keys(source);
   return collection.filter(function (obj) {
    for(var i = 0; i < srcKeys.length; i++) {
      if(!obj.hasOwnProperty(srcKeys[i]) || obj[srcKeys[i]] !== source[srcKeys[i]]) {
        return false;
      }
    }
    return true;
  });
}
```

HOẶC

```
return collection.filter(function (obj) {
    return srcKeys.every(function (key) {
      return obj.hasOwnProperty(key) && obj[key] === source[key];
    });
  });
```

Câu 9.5. {Spinal Tap Case}

- **Đề bài:** Chuyển câu thành một xâu mà các ký tự đều ở dạng chữ thường, các từ trong câu được ngăn cách bởi dấu "-".
  Ví dụ: "A b C" -> "a-b-c"; "aBC" -> "a-b-c"
- Ý tưởng:
  - Thay thế các "aB" thành "a B".
  - Chia câu thành các từ.
  - Chuyển chúng về chữ thường.
  - Nối chúng bằng dấu -.
- **{Đáp án}**
  str.replace(/([a-z])([A-Z])/g, "$1 $2").split(/?:_| /).join("-").toLowerCase();
- **{Lưu ý}**
  - Khi sử dụng replace để thay đổi các regex, cần sử dụng dấu () xung quanh đối tượng muốn tham chiếu đến sau này.
  - Sử dụng cờ g để match hết toàn bộ các pattern mà mình cần.
- **Đáp án*:**

```
function spinalCase(str) {
  var regex = /\s+|_+/g;
  str = str.replace(/([a-z])([A-Z])/g, '$1 $2');
  return str.replace(regex, '-').toLowerCase();
```

```
}
```

HOẶC

```
str.split(/\s|_|(?=[A-Z])/).join('-').toLowerCase()
```

Câu 9.6. {Pig Latin}

- **Đề bài:** Chuyển câu về dạng Pig Latin. Pig Latin lấy phụ âm/cụm phụ âm đầu tiên của một từ Tiếng Anh, chuyển nó về cuối từ đó và thêm ay vào sau cùng. Nếu từ bắt đầu bằng nguyên âm thì thêm way vào cuối từ.
- Ví dụ: california -> aliforniacay; alaska -> alaskaway
- **{Ý tưởng*}**
    - Sử dụng test() để kiểm tra xem liệu xâu có bắt đầu bằng một nguyên âm hay không. (Sử dụng regex: /^ueoai/).
    - Nếu là nguyên âm, thêm "way" vào cuối từ.
    - Ngược lại, tìm vị trí của nguyên âm đầu tiên trong từ.
        - Nếu từ không có nguyên âm, thêm "ay" vào cuối từ.
        - Nếu có, sử dụng slice() để cắt từ từ vị trí đầu đến vị trí nguyên âm, replace() cụm phụ âm (/[^ueoai]+/) bằng "". Thêm đoạn này + "ay" vào cuối từ.
- **{Đáp án*}**

```
let vowelRegex = /^[ueoai]/;
if(vowelRegex.test(str)) {
        return str + "way";
} else {
        let vowel = str.match(/[ueoai]/);
        if(vowel == null) {
                return str + "ay";
        } else {
                let newStr = str.slice();
                let consonant = str.slice(0, vowel.index);
                return newStr.replace(/[^ueoai]+/, "") + consonant + "ay";
        }
}
```

- Đáp án:

```
function translatePigLatin(str) {
  var pigLatin = '';
  var regex = /[aeiou]/gi;
  if (str[0].match(regex)) {
    pigLatin = str + 'way';
  } else if(str.match(regex) === null) {
    pigLatin = str + 'ay';
  } else {
    var vowelIndice = str.indexOf(str.match(regex)[0]);
    pigLatin = str.substr(vowelIndice) + str.substr(0, vowelIndice) + 'ay';
  }
  return pigLatin;
}
```

Câu 9.7. {Search and Replace}

- **Đề bài:** Tìm kiếm và thay thế một từ trong câu.
- Lưu ý: Khi thay thế một từ bằng một từ khác, cần đảm bảo rằng tính hoa/ thường của chữ cái đầu tiên trong từ mới giống với từ cũ. Ví dụ: Khi thay thế Abc bằng def thì def cần được chuyển thành Def.
- **{Ý tưởng*}**
    - Chia ra 2 trường hợp: Khi chữ cái bắt đầu của before là chữ hoa hoặc bằng chữ thường.

- Sử dụng replace() để thay thế chữ đầu tiên của after ([^[a-zA-Z]]) thành chữ hoa/ thường tương ứng. (Lưu ý, toLowerCase() và toUpperCase() không làm thay đổi xâu gốc nên cần lưu ký tự được chuyển ra một biến khác).
- Thay thế before bằng newAfter.

- **{Đáp án*}**

```
function myReplace(str, before, after) {
    var newAfter = "";
    var first = after[0];
    if (/^[A-Z]/.test(before)) {
        newAfter = after.replace(/(^[a-zA-Z])/, first.toUpperCase());
    } else {
        newAfter = after.replace(/(^[a-zA-Z])/, first.toLowerCase());
    }
    return str.replace(before, newAfter);
}
```

- Đáp án:

```
function myReplace(str, before, after) {
  var index = str.indexOf(before);
  if (str[index] === str[index].toUpperCase()) {
    after = after.charAt(0).toUpperCase() + after.slice(1);
  }
  str = str.replace(before, after);
  return str;
}
```

Câu 9.8. {DNA Pairing}

- **Đề bài:** Với các cặp gen cơ bản là A - T và C - G, cho một NST (một xâu bao gồm các ký tự C, A, T, G), tìm gen tương ứng của các gen trong NST này và trả về mảng 2 chiều các cặp gen.
  Ví dụ: CAT -> [["C", "G"], ["A", "T"], ["T", "A"]];
- **{Ý tưởng*}**
  - Chia xâu thành mảng các ký tự.
  - Sử dụng map() trên mảng này. (Trong hàm callback, trả về kết quả trong từng trường hợp khi ký tự đầu vào lần lượt là A, T, C, G)
- **{Đáp án*}**

```
function pairElement(str) {
    let chars = str.split("");
    return chars.map(function(x) {
        if(x == "C") return ["C", "G"];
        else if(x == "G") return ["G", "C"];
        else if(x == "A") return ["A", "T"];
        else return ["T", "A"];
    });
}
pairElement("GCG");
```

- Đáp án:

```
function pairElement(str) {
    var pairs = {
      "A": "T","T": "A","C": "G","G": "C"
    }
    var arr = str.split("");
    return arr.map(x => [x,pairs[x]]);
  }
```

Câu 9.9. {Missing letters}

- **Đề bài:** Tìm ký tự bị mất trong một dãy các ký tự. Trả về underfined nếu không có ký tự nào bị mất.
  Ví dụ: abd -> c; abc -> underfined**.**

- **{Ý tưởng*}**
  - Tìm mã ASCII của phần tử đầu tiên trong dãy (str.charCodeAt(index)).
  - Sử dụng map (chú ý, tham số thứ 2 của hàm callback được dùng cho index) để tìm mã ASCII của phần tử bị mất (bằng cách tăng biến đếm).
  - Sử dụng String.fromCharCode() để trả về phần tử bị mất.
- **{Lưu ý}**

  Khai báo tổng quát của map():

  let new_arr = arr.map(function callback(currentValue[, index[, array]])

  Trong đó: currentValue là giá trị phần tử đang được xử lý, index là chỉ số của phần tử đang được xử lý còn array là mảng gọi đến map().

  Khai báo tổng quát của filter() HOÀN TOÀN TƯƠNG TỰ.
- Đáp án:

```
function fearNotLetter(str) {
  for(var i = 0; i < str.length; i++) {
    var code = str.charCodeAt(i);
    if (code !== str.charCodeAt(0) + i) return String.fromCharCode(code - 1);
  }
  return undefined;
}
```

Câu 9.10. {Sorted Union}

- **Đề bài:** Cho 2 hoặc nhiều mảng. Trả về mảng mới được hợp thành từ các mảng cũ, trong đó, các phần tử chỉ xuất hiện 1 lần và theo thứ tự.

  Ví dụ: [1, 2], [3, 2, 1], [1] => [1, 2, 3]**.**
- **{Ý tưởng*}**
  - Sử dụng arguments để duyệt hết các tham số được truyền vào hàm. Sử dụng concat() để nối các mảng con thành một mảng duy nhất.
  - Trên mảng mới này, sử dụng filter() để lọc ra mảng cần tìm. Lưu ý, hàm comeback của filter() sử dụng cả 3 tham số để kiểm tra xem giá trị của phần tử đang xét đã xuất hiện trước đó trong mảng hay chưa.
- **{Đáp án}**

```
function uniteUnique(arr) {
        let newArr = [];
        for(let i = 0; i < arguments.length; i++)
                newArr = newArr.concat(arguments[i]);
        let resultArr = newArr.filter(function(element, index, newArr) {
                for(let i = 0; i < index; i++) ;
                if(element == newArr[i])
                        return false;
                        return true;
        });
        return resultArr;
}
```

- Đáp án:

```
function uniteUnique(arr) {
  var args = [...arguments];
  var result = [];
  for(var i = 0; i < args.length; i++) {
    for(var j = 0; j < args[i].length; j++) {
       if(!result.includes(args[i][j])) {
        result.push(args[i][j]);
      }
    }
  }
  return result;
```

```
}
```

Câu 3.11. {ConvertHTML}

- **Đề bài:** Chuyển các ký tự &, ', ", <, > về mã của chúng.
  Ví dụ: 'A&B' -> 'A&B'
- **{Ý tưởng*}**
  Sử dụng một dãy replace() liên tiếp thay thế các ký hiệu bằng mã tương ứng của chúng.
- **{Đáp án}**
- Đáp án:

```
function convertHTML(str) {
      htmlEntities = {
        '&':'&amp;',
        '<':'&lt;',
        '>':'&gt;',
        '"':'&quot;',
        '\'':"&apos;"
      };
      return str.split('').map(entity => htmlEntities[entity] || entity).join('');
 }
```

Câu 3.12. {Sum All Odd Fibonacci Numbers}

- **Đề bài:** Tính tổng các số Fibonaci lẻ nhỏ hơn một số cho trước.
  Ví dụ: 4 -> 1 + 1 + 3 = 5
- **{Ý tưởng*}**
  - Khởi tạo mảng gồm F0 và F1. Sau mỗi vòng lặp (so sánh tổng F0 và F1 với số cho ở đề bài, ta thêm F0 + F1 vào mảng.
  - Sử dụng filter() để lọc ra các phần tử lẻ trong mảng.
  - Sử dụng reduce() để tính tổng.
- **{Đáp án}**
- Đáp án:

```
function sumFibs(num) {
    var prevNumber = 0;
    var currNumber = 1;
    var result = 0;
    while (currNumber <= num) {
        if (currNumber % 2 !== 0) result += currNumber;
        currNumber += prevNumber;
        prevNumber = currNumber - prevNumber;
    }

    return result;
}
```

Câu 3.13. {Sum All Primes}

- **Đề bài:** Tính tổng các số nguyên tố nhỏ hơn hoặc bằng một số cho trước.
  Ví dụ: 10 -> 2 + 3 + 5 + 7 = 17
- Đáp án:

```
function sumPrimes(num) {
  let arr = Array.from({length: num+1}, (v, k) => k).slice(2);
  let onlyPrimes = arr.filter((n) => {
    let m = n-1;
    while (m > 1 && m >= Math.sqrt(n)) {
      if ((n % m) === 0)
        return false;
        m--;
    }
      return true;
  });
```

```
  return onlyPrimes.reduce((a,b) => a + b);
}
```

Câu 3.14. {Smallest Common Multiple}

```
// Tìm bội chung nhỏ nhất.
function smallestCommons(arr) {
    var range = [];
    for (var i = Math.max(arr[0], arr[1]); i >= Math.min(arr[0], arr[1]); i--) {
    range.push(i);
    }
    var lcm = range[0];
    for (i = 1; i < range.length; i++) {
             var GCD = gcd(lcm, range[i]);
             lcm = (lcm * range[i]) / GCD;
    }
    return lcm;

    function gcd(x, y) {    // Implements the Euclidean Algorithm
             if (y === 0) return x;
             else return gcd(y, x%y);
    }
}
```

Câu 3.15. {Drop it}

```
// Xóa phần tử của mảng thỏa mãn điều kiện cho trước.
function dropElements(arr, func) {
  while(arr.length > 0 && !func(arr[0])) {
    arr.shift();
  }
  return arr;
}
dropElements([1, 2, 3], function(n) {return n < 3; });
```

Câu 3.16. {Steamroller}

```
// Ghép các mảng lồng nhau.
function steamrollArray(arr) {
       let flat = [].concat(...arr);
       return flat.some(Array.isArray) ? steamrollArray(flat) : flat;
}
```

Câu 3.17. {Binary Agents}

```
// Chuyển sang nhị phân.
function binaryAgent(str) {
       return String.fromCharCode(...str.split(" ").map(function(char){ return
parseInt(char, 2); }));
}
```

Câu 3.18. {Everything Be True}

```
function truthCheck(collection, pre) {
       return collection.every(function (element) {
             return element.hasOwnProperty(pre) && Boolean(element[pre]);
       });
}
```

Câu 3.19. {Arguments Optional}

```
function addTogether() {
       var checkNum = function(num) {
             if (typeof num !== 'number') return undefined;
             else return num;
       };
```

```
            if (arguments.length > 1) {
                  var a = checkNum(arguments[0]);
                  var b = checkNum(arguments[1]);
                  if (a === undefined || b === undefined) return undefined;
                  else return a + b;

            } else {
                  var c = arguments[0];
                  if (checkNum(c)) {
                        return function(arg2) {
                              if (c === undefined || checkNum(arg2) === undefined)
                                    return undefined;
                              else return c + arg2;
                        };
                  }
            }
      }
```

Câu 3.20. {Make a Person}

```
      var Person = function(firstAndLast) {
        var fullName = firstAndLast;
        this.getFirstName = function() {
          return fullName.split(" ")[0];
        };
        this.getLastName = function() {
          return fullName.split(" ")[1];
        };
        this.getFullName = function() {
          return fullName;
        };
        this.setFirstName = function(name) {
          fullName = name + " " + fullName.split(" ")[1];
        };
        this.setLastName = function(name) {
          fullName = fullName.split(" ")[0] + " " + name;
        };
        this.setFullName = function(name) {
          fullName = name;
        };
      };
```

Câu 3.21. {Map the Debris}

```
      [{name : "sputnik", avgAlt : 35873.5553}] should return [{name: "sputnik",
orbitalPeriod: 86400}].
      function orbitalPeriod(arr) {
        var GM = 398600.4418;
        var earthRadius = 6367.4447;
        var a = 2 * Math.PI;
        var newArr = [];
        var getOrbPeriod = function(obj) {
          var c = Math.pow(earthRadius + obj.avgAlt, 3);
          var b = Math.sqrt(c / GM);
          var orbPeriod = Math.round(a * b);
          delete obj.avgAlt;
          obj.orbitalPeriod = orbPeriod;
          return obj;
        };
```

```
        for (var elem in arr) {
          newArr.push(getOrbPeriod(arr[elem]));
        }

        return newArr;
      }
```

Câu 3.22. {Palindrome Checker}

```
  function palindrome(str) {
      return str.replace(/[\W_]/g, '').toLowerCase() ===
             str.replace(/[\W_]/g, '').toLowerCase().split('').reverse().join('');
    }
```

Câu 3.22. {Roman Numeral Converter}

```
var convertToRoman = function(num) {
  var decimalValue = [ 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 ];
  var romanNumeral = [ 'M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I' ];
  var romanized = '';
  for (var index = 0; index < decimalValue.length; index++) {
    while (decimalValue[index] <= num) {
      romanized += romanNumeral[index];
      num -= decimalValue[index];
    }
  }
  return romanized;}
```

Câu 3.23. {Caesars Cipher}

```
    function rot13(str) {
      var rotCharArray = [];
      var regEx = /[A-Z]/ ;
      str = str.split("");
      for (var x in str) {
        if (regEx.test(str[x]))
    rotCharArray.push((str[x].charCodeAt() - 65 + 13) % 26 + 65);
        else rotCharArray.push(str[x].charCodeAt());
      }
      str = String.fromCharCode.apply(String, rotCharArray);
      return str;
    }
```

Câu 3.24. {Telephone Number Validator}

```
function telephoneCheck(str) {
   var regex = /^(1\s?)?(\(\d{3}\)|\d{3})[\s\-]?\d{3}[\s\-]?\d{4}$/;
   return regex.test(str);
}
```

Câu 3.25. {Cash Register}

```
// Create an array of objects which hold the denominations and their values
var denom = [
  { name: 'ONE HUNDRED', val: 100.00},{ name: 'TWENTY', val: 20.00},
  { name: 'TEN', val: 10.00},name: 'FIVE', val: 5.00},
  { name: 'ONE', val: 1.00},{ name: 'QUARTER', val: 0.25},
  { name: 'DIME', val: 0.10},{ name: 'NICKEL', val: 0.05},
  { name: 'PENNY', val: 0.01}
];
```

```
function checkCashRegister(price, cash, cid) {
  var output = { status: null, change: [] };
  var change = cash - price;
  // Transform CID array into drawer object
  var register = cid.reduce(function(acc, curr) {
    acc.total += curr[1];
    acc[curr[0]] = curr[1];
    return acc;
  }, { total: 0 });

  // Handle exact change
  if (register.total === change) {
    output.status = 'CLOSED';
    output.change = cid;
    return output;
  }

  // Handle obvious insufficient funds
  if (register.total < change) {
    output.status = 'INSUFFICIENT_FUNDS';
    return output;
  }

  // Loop through the denomination array
  var change_arr = denom.reduce(function(acc, curr) {
    var value = 0;
    // While there is still money of this type in the drawer
    // And while the denomination is larger than the change remaining
    while (register[curr.name] > 0 && change >= curr.val) {
      change -= curr.val;
      register[curr.name] -= curr.val;
      value += curr.val;

      // Round change to the nearest hundreth deals with precision errors
      change = Math.round(change * 100) / 100;
    }
    // Add this denomination to the output only if any was used.
    if (value > 0) {
        acc.push([ curr.name, value ]);
    }
    return acc; // Return the current change_arr
  }, []); // Initial value of empty array for reduce

  // If there are no elements in change_arr or we have leftover change, return
  // the string "Insufficient Funds"
  if (change_arr.length < 1 || change > 0) {
    output.status = 'INSUFFICIENT_FUNDS';
    return output;
  }

  // Here is your change, ma'am.
  output.status = 'OPEN';
  output.change = change_arr;
  return output;
}
```

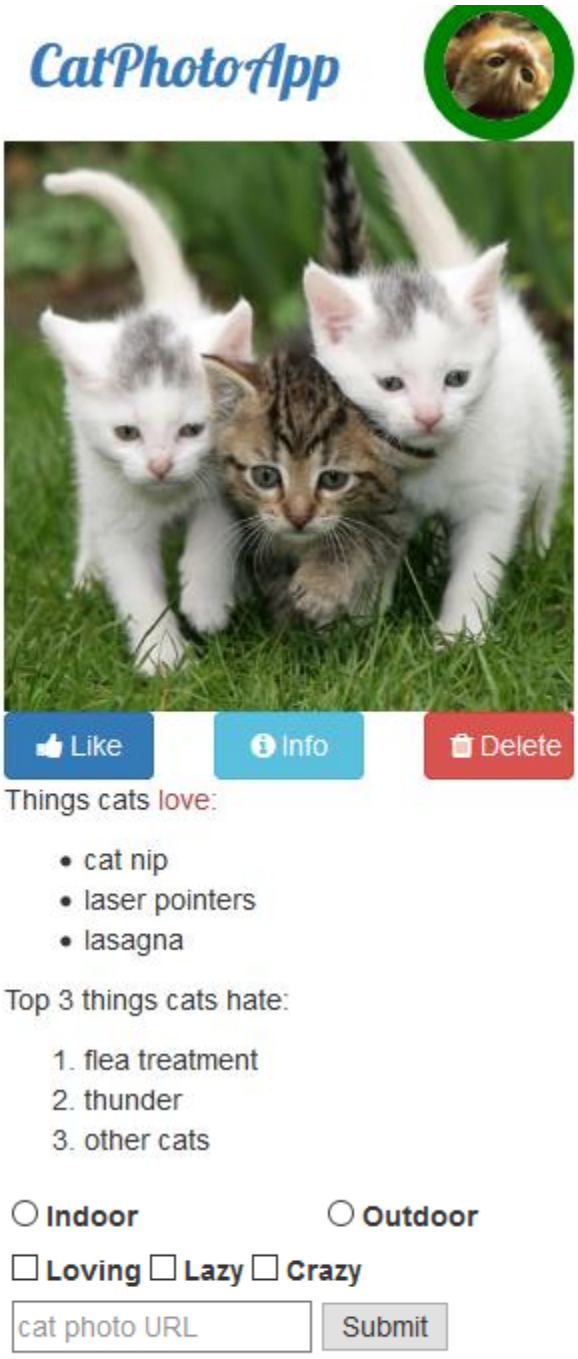
```
checkCashRegister(19.50, 20.00, [["PENNY", 1.01], ["NICKEL", 2.05], ["DIME", 3.10], ["QUARTER",
4.25], ["ONE", 90.00], ["FIVE", 55.00], ["TEN", 20.00], ["TWENTY", 60.00], ["ONE HUNDRED", 100.00]]);
```

II. Front End Libraries Certification

1. Bootstrap

- **Bootstrap** là một framework miễn phí, mã nguồn mở, được dùng để tương thích giao diện web với các thiết bị có kích thước màn hình khác nhau.
- **Kết quả code.**

1.1. Ứng dụng ảnh Mèo




```
<style>
  h2 {font-family: Lobster, Monospace; }
  .thick-green-border {
    border-color: green; border-width: 10px; -style: solid; border-radius: 50%;
  }
</style>
```

***; Câu 1.1. Thêm thẻ ***

```
<div class="container-fluid">
```

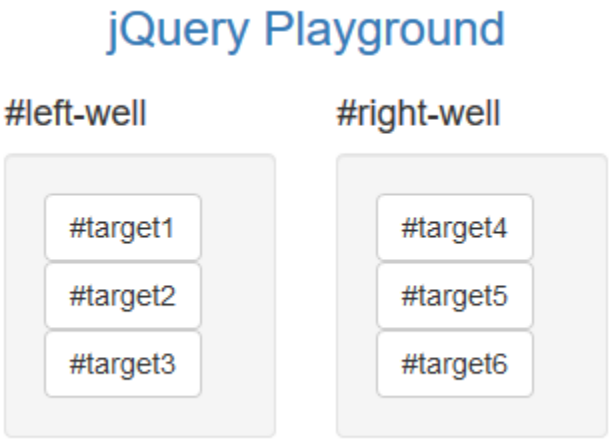
; Câu 1.11. Đặt tiêu đề và hình trên cùng một hàng bằng và .

```
  <div class="row">
```

```
    ; Câu 1.3. Căn chỉnh giữa, thêm class = “text-center”.
    <div class="col-xs-8"><h2 class="text-primary text-center">CatPhotoApp</h2>
    </div>
    <div class="col-xs-4">
      <a href="#"><img class="img-responsive thick-green-border"
src="https://bit.ly/fcc-relaxing-cat" alt="A cute orange cat lying on its back."></a>
    </div>
  </div>
  ; Câu 1.2. Điều chỉnh kích thước ảnh phù hợp với kích thước màn hình điện thoại.
Thêm class = img-responsive.
  <img src="https://bit.ly/fcc-running-cats" class="img-responsive" alt="Three
kittens running towards the camera.">
  ; Câu 1.9. Đặt các button vào cùng một hàng bằng cách bổ sung <div class=“row”> và
<div class=“col-xs-4”> (Một hàng chứa 12 cột, 3 button mỗi button chiếm 4 cột).
  <div class="row">
    <div class="col-xs-4">
      ; Câu 1.12. Thêm icon bằng thẻ <i></i>.
      <button class="btn btn-block btn-primary"><i class="fa fa-thumbs-up"></i>
Like</button>
    </div>
    <div class="col-xs-4">
      ; Câu 1.7. Thêm class = “btn-info” để thiết lập màu cho Info.
      <button class="btn btn-block btn-info"><i class="fa fa-info-circle"></i>
Info</button>
    </div>
    <div class="col-xs-4">
      ; Câu 1.8. Thêm class = “btn-danger” để thiết lập màu cho Delete.
      <button class="btn btn-block btn-danger"><i class="fa fa-trash"></i>
Delete</button>
    </div>
  </div>
  ; Câu 1.10. Sử dụng <span> để đặt các thành phần khác nhau trên cùng một hàng.
  <p>Things cats <span class="text-danger">love:</span></p>
  <ul>
    <li>cat nip</li>
    <li>laser pointers</li>
    <li>lasagna</li>
  </ul>

  <p>Top 3 things cats hate:</p>
  <ol>
    <li>flea treatment</li>
    <li>thunder</li>
    <li>other cats</li>
  </ol>

  <form action="/submit-cat-photo">
    <div class="row">
```

```
        <div class="col-xs-6">
          <label><input type="radio" name="indoor-outdoor"> Indoor</label>
        </div>
        <div class="col-xs-6">
          <label><input type="radio" name="indoor-outdoor"> Outdoor</label>
        </div>
      </div>
      <div class="row">
        <div class="col-xs-4">
          <label><input type="checkbox" name="personality"> Loving</label>
        </div>
        <div class="col-xs-4">
          <label><input type="checkbox" name="personality"> Lazy</label>
        </div>
        <div class="col-xs-4">
          <label><input type="checkbox" name="personality"> Crazy</label>
        </div>
      </div>

      <input type="text" class="form-control" placeholder="cat photo URL" required>
      ; Câu 1.4. Thêm button trong Bootstrap.
      ; Câu 1.5. Thêm class = "btn-block" để button điều chỉnh kích thước theo chiều rộng của trang.
      ; Câu 1.6. Thêm class = "btn-primary" để thiết lập màu cho Submit.

      <button type="submit" class="btn btn-primary"><i class="fa fa-paper-plane"></i>
Submit</button>
    </form>
</div>
```

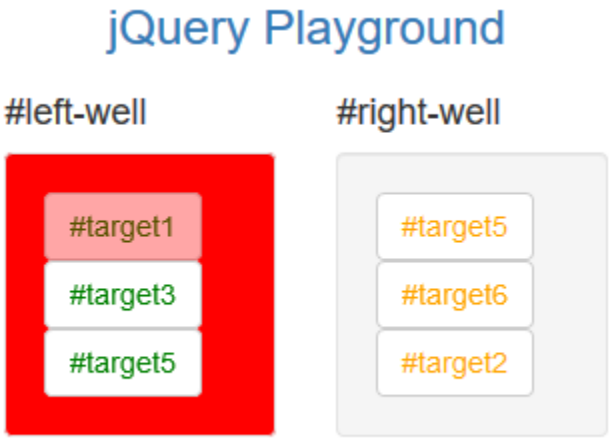
1.2. Ứng dụng jQuery




```
<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
```

```
        ; Câu 1.13. Well là một lớp thường được sử dụng trong Bootstrap.
        <div class="well" id="left-well">
          ; Câu 1.14. Target được sử dụng để hỗ trợ các truy vấn trong jQuery.
          <button class="btn btn-default target" id="target1">#target1</button>
          <button class="btn btn-default target" id="target2">#target2</button>
          <button class="btn btn-default target" id="target3">#target3</button>
        </div>
      </div>
      <div class="col-xs-6">
        <h4>#right-well</h4>
        <div class="well" id="right-well">
          <button class="btn btn-default target" id="target4">#target4</button>
          <button class="btn btn-default target" id="target5">#target5</button>
          <button class="btn btn-default target" id="target6">#target6</button>
        </div>
      </div>
    </div>
  </div>
```

2. **jQuery**




```
// Câu 2.1. Thêm thẻ script và $(document).ready(function() {});
<script>
  $(document).ready(function() {
    // Câu 2.2. Thêm các lớp animated và bounce vào các phần tử button.
    $("button").addClass("animated bounce");
    // Câu 2.3. Thêm các lớp animated và shake vào các phần tử thuộc lớp well.
    $(".well").addClass("animated shake");
    // Câu 2.4. Thêm các lớp animated và fadeOut để làm #target3 mờ dần.
    $("#target3").addClass("animated fadeOut");
    // Câu 2.5. Loại bỏ lớp btn-default ra khỏi các phần tử button.
    $("button").removeClass("btn-default");
    // Câu 2.6. Sửa đổi thuộc tính color của #target1 sang "red".
    $("#target1").css("color", "red");
    // Câu 2.7. Không cho phép click vào một button.
    $("button").prop("disabled", true);
```

```
    // Câu 2.8. Thay đổi nội dung #target7, <em></em> cho phép in nghiêng văn bản.
    $("#target4").html("<em>#target4</em>");
    // Câu 2.9. Xóa #target4.
    $("#target4").remove();
    // Câu 2.10. Đính #target2 sang #right-well.
    $("#target2").appendTo("#right-well");
    // Câu 2.11. Tạo clone của #target5 và đính vào #left-well.
    $("#target5").clone().appendTo("#left-well");
    // Câu 2.12. Thay đổi màu nền phần tử cha của #target1.
    $("#target1").parent().css("background-color","red");
    // Câu 2.13. Thay đổi màu nền phần tử con của #right-well.
    $("#right-well").children().css("color","orange")
    // Câu 2.14. Thêm các lớp animated và bounce vào phần tử thứ 2 của nhóm các phần tử có lớp target
.
    $(".target:nth-child(2)").addClass("animated bounce");
    // Câu 2.15. Thêm các lớp animated và shake vào các phần tử chẵn của nhóm các phần tử có lớp
target (Bắt đầu từ 0).
    $(".target:even").addClass("animated shake");
    // Câu 2.16. Thêm các lớp animated và shake vào các phần tử lẻ của nhóm các phần tử có lớp target.
    $(".target:odd").addClass("animated shake");
    // Câu 2.17. Thêm các lớp animated và hinge vào phần tử body.
    $("body").addClass("animated hinge");
});
</script>

<div class="container-fluid">
  <h3 class="text-primary text-center">jQuery Playground</h3>
  <div class="row">
    <div class="col-xs-6">
      <h4>#left-well</h4>
      <div class="well" id="left-well">
        <button class="btn btn-default target" id="target1">#target1</button>
        <button class="btn btn-default target" id="target2">#target2</button>
        <button class="btn btn-default target" id="target3">#target3</button>
      </div>
    </div>
    <div class="col-xs-6">
      <h4>#right-well</h4>
      <div class="well" id="right-well">
        <button class="btn btn-default target" id="target4">#target4</button>
        <button class="btn btn-default target" id="target5">#target5</button>
        <button class="btn btn-default target" id="target6">#target6</button>
      </div>
    </div>
  </div>
</div>
```

3. Sass

HTML

```
<h1 class="header">Learn Sass</h1>
```

```
<div class="blog-post">
  <h2>Some random title</h2>
  <p>This is a paragraph with some random text in it</p>
</div>
<div class="blog-post">
  <h2>Header #2</h2>
  <p>Here is some more random text.</p>
</div>
<div class="blog-post">
  <h2>Here is another header</h2>
  <p>Even more random text within a paragraph</p>
</div>
```

SASS

Câu 3.1. {Khai báo biến $text-color}

```
<style type='text/sass'>
  $text-color: red;
  .header{text-align: center;}
  .blog-post, h2 {color: $text-color;}
</style>
```

Câu 3.2. {Khai báo style lồng nhau}

```
<style type='text/sass'>
  .blog-post {
      h1 {text-align: center; color: blue;}
      p {font-size: 20px;}
  }
</style>
```

Câu 3.3. {Mixins – Hàm trong CSS}

```
<style type='text/sass'>
  @mixin border-radius($radius){
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius;
    border-radius: $radius;
  }
  #awesome {
    width: 150px;
    height: 150px;
    background-color: green;
    @include border-radius(15px);
  }
</style>
```

Câu 3.4. {@if - @else}

```
<style type='text/sass'>
  @mixin border-stroke($val) {
    @if $val == light{
       border: 1px solid black;
    }
    @else if $val == medium {
       border: 3px solid black;
```

```
        }
        @else if $val == heavy {
           border: 6px solid black;
        }
        @else {
           border: none;
        }
      }

      #box {
        width: 150px;
        height: 150px;
        background-color: red;
        @include border-stroke(medium);
      }
    </style>
```

Câu 3.5. {@for}

```
    @for $j from 1 through 6 {
        .text-#{$j}{font-size: $j*10px;}
    }
```

Câu 3.6. {@each}

```
    // LIST
    @each $color in blue, red, green {
      .#{$color}-text {color: $color;}
    }
    // MAP
    $colors: (color1: blue, color2: red, color3: green);
    @each $key, $color in $colors {
      .#{$color}-text {color: $color;}
    }
```

Câu 3.7. {@while}

```
    @while $x < 11 {
          .text-#{$x} {font-size: $x*5;}
          $x: $x + 1;
    }
```

Câu 3.8. {@import}

```
    @import 'variables'                          ; Bổ sung file _variables.scss
```

Câu 3.9. {@extend – Tương tự kế thừa}

```
    h3{text-align: center;}
    .info{
          width: 200px;
          border: 1px solid black;
          margin: 0 auto;
    }
    .info-important{
          @extend .info;
          background-color: magenta;
    }
```

III. Data Visualization Certification

1. Data Visualization with D3

Câu 1.1. {Giới thiệu}

- D3 = Data Driven Documents, là một thư viện JS được sử dụng để hình ảnh hóa dữ liệu trong trình duyệt.
- Đầu vào: Dữ liệu ở nhiều định dạng khác nhau. Đầu ra: Hình ảnh của chúng.

Câu 1.2. {Thêm phần tử}

```
// Chọn phần tử body, thêm vàtheeos h1, thay đổi nội dung của h1 thành “Learning D3”.
<body>
	<script>
		d3.select("body").append("h1").text("Learning D3");
	</script>
</body>

// Chọn nhiều phần tử với selectAll().
<body>
	<ul>
		<li>Example</li>
		<li>Example</li>
		<li>Example</li>
	</ul>

	<script>
		d3.selectAll("li").text("list item");
	</script>
</body>
```

Câu 1.3. {Làm việc với dữ liệu}

```
<body>
	<ul></ul>
	<script>
		const dataset = ["a", "b", "c"];
		d3.select("ul").selectAll("li")
		.data(dataset)            ; Chọn phần tử trên DOM để đính dữ liệu vào.
		.enter()                  ; Tạo một phần tử ứng với mỗi phần dữ liệu
                                  trong tập.
		.append("li")
		.text("New item");
	</script>
</body>
```

Câu 1.4. {Dữ liệu động, style()}

```
d3.select("body").selectAll("h2")
.data(dataset)
.enter()
.append("h2")
.text((d) => d + " USD");             ; Tạo phần tử với trường text nhận giá trị là
                                      các phần tử trong tập dữ liệu
.style("font-family", "verdana");     ; Thêm style cho từng phần tử.
```

Câu 1.4. {attr()}

```
<style>
	.bar {
		width: 25px;
		height: 100px;
		display: inline-block;
		background-color: blue;
	}
</style>
```

```
<body>
	<script>
		const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];
		d3.select("body").selectAll("div")
		.data(dataset)
		.enter()
		.append("div")
		.attr("class", "bar");	; Thay đổi thuộc tính (cụ thể là thêm class) cho các phần tử mới.
	</script>
</body>
```

Câu 1.5. {Bài tập}

```
// Nội dung: Thay đổi chiều cao của các phần tử theo giá trị dữ liệu trong tập dữ liệu.

d3.select("body").selectAll("div")
	.data(dataset)
	.enter()
	.append("div")
	.attr("class", "bar")
	.style("height", (d) => d + "px");
```

Câu 1.6. {Bài tập}

```
// Nội dung: Thay đổi biểu diễn đồ thị cột.
// Cụ thể: 1.Thêm khoảng trắng giữa các cột (bằng cách thêm margin vào lớp bar).
	2.Tăng chiều cao các cột bằng cách nhân với cùng một giá trị nào đấy.

<style>
	.bar {
		width: 25px;
		height: 100px;
		margin: 2px;
		display: inline-block;
		background-color: blue;
	}
	</style>
	<body>
		<script>
			const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];
			d3.select("body").selectAll("div")
			.data(dataset)
			.enter()
			.append("div")
			.attr("class", "bar")
			.style("height", (d) => (10*d + "px"));
		</script>
	</body>
```

Câu 1.7. {SVG}

```
// Sử dụng <svg></svg> để tạo hình khối cơ bản.
<style>
	svg {
		background-color: pink;
	}
</style>
<body>
	<script>
```

```
            const dataset = [12, 31, 22, 17, 25, 18, 29, 14, 9];
            const w = 500;
            const h = 100;
            const svg = d3.select("body")
            .append("svg")
            .attr("width", w)
            .attr("height", h)
        </script>
</body>
```

// Thêm hình chữ nhật

```
const svg = d3.select("body")
        .append("svg")
        .attr("width", w)
        .attr("height", h)
        .append("rect")
        .attr("width", 25)
        .attr("height", 100)
        .attr("x", 0)
        .attr("y", 0);
```

// Kết hợp với các câu lệnh khác

```
const svg = d3.select("body")
.append("svg")
.attr("width", w)
.attr("height", h);

svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", 0)
.attr("y", 0)
.attr("width", 25)
.attr("height", 100);
```

Câu 1.8. {Bài tập}

// Nội dung: Thay đổi tọa độ các cột trong biểu đồ cột (Tránh chồng lên nhau).
// Cụ thể: + Đối với biểu đồ cột, các cột cần dóng hàng theo chiều ngang. Tức, y là bằng nhau và bằng 0. + Giá trị x cần thay đổi.

```
const svg = d3.select("body")
.append("svg")
.attr("width", w)
.attr("height", h);

svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", (d, i) => {
        return i*30;
})
.attr("y", 0)
.attr("width", 25)
.attr("height", 100);
```

// Nội dung: Thay đổi độ cao các cột tương ứng với giá trị dữ liệu.
*// Đáp án: Thay .attr("height", 100) thành .attr("height", (d, i) => 3*d).*

// Nội dung: Lật ngược biểu đồ (do biểu đồ đang bị ngược).
*// Đáp án: Thay .attr("y", 0) thành .attr("y", (d, i) => h - 3*d), trong đó h là độ cao của svg.*

// Nội dung: Thay đổi màu của các cột.
// Đáp án: Thêm .attr("fill", "navy");

Câu 1.9. {Bài tập - Nhãn}

// Nội dung: Thêm nhãn cho các phần tử.
// Cụ thể: Sử dụng phần tử text() trong SVG. Lưu ý, các text này cũng cũng có tọa độ x, y.

```
svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.attr("x", (d, i) => i * 30)
.attr("y", (d, i) => h - 3 * d - 3)
.text((d) => d);
```

// Nội dung: Thêm style cho nhãn.
// Đáp án: Thêm .style("fill","red").style("font-size", "25px");.

Câu 1.10. {Bài tập – Sử dụng hover để thay đổi màu cột khi trỏ đến}

```
<style>
	.bar:hover {
		fill: brown;
}
</style>
-----------------------------------
svg.selectAll("rect")
.data(dataset)
.enter()
.append("rect")
.attr("x", (d, i) => i * 30)
.attr("y", (d, i) => h - 3 * d)
.attr("width", 25)
.attr("height", (d, i) => 3 * d)
.attr("fill", "navy")
.attr("class", "bar");
```

Câu 1.11. {Bài tập – Biểu đồ điểm phân tán}

```
<body>
	<script>
		const dataset = [
		[ 34, 78 ],
		[ 109, 280 ],
		[ 310, 120 ],
		[ 79, 411 ],
		[ 420, 220 ],
		[ 233, 145 ],
		[ 333, 96 ],
```

```
                [ 222, 333 ],
                [ 78, 320 ],
                [ 21, 123 ]
                ];

                const w = 500;
                const h = 500;

                const svg = d3.select("body")
                .append("svg")
                .attr("width", w)
                .attr("height", h);

                svg.selectAll("circle")
                .data(dataset)
                .enter()
                .append("circle")
                .attr("cx", (d) => d[0])
                .attr("cy", (d) => h - d[1])
                .attr("r", 5);
        </script>
    </body>
```

// Nội dung: Thêm nhãn cho các điểm.

```
svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.attr("x", (d, i) => d[0]+ 5)
.attr("y", (d, i) => h - d[1])
.text((d) => d[0] + ", " + d[1]);
```

Câu 1.12. {Bài tập – Thay đổi scale}

```
const scale = d3.scaleLinear();        // Create the scale here
const output = scale(50);              // Call the scale with an argument here
```

// Nội dung: Thay đổi scale từ domain-> range.

```
const scale = d3.scaleLinear();
scale.domain([250, 500]);
scale.range([10,150]);
```

// Nội dung: Tìm min, max của tập dữ liệu.

```
const positionData = [[1, 7, -4],[6, 3, 8],[2, 8, 3]]
const output = d3.max(positionData, (d) => d[2]);
```

// Nội dung: Thay đổi scale theo tham số.

```
const padding = 30;

const xScale = d3.scaleLinear()
.domain([0, d3.max(dataset, (d) => d[0])])
.range([padding, w - padding]);

const yScale = d3.scaleLinear()
.domain([0, d3.max(dataset, (d) => d[1])])
.range([h - padding, padding]);
```

```
// Nội dung: Thay đổi tọa độ các hình theo scale.
svg.selectAll("circle")
.data(dataset)
.enter()
.append("circle")
.attr("cx", (d) => xScale(d[0]))
.attr("cy", (d) => yScale(d[1]))
.attr("r", 5);
svg.selectAll("text")
.data(dataset)
.enter()
.append("text")
.text((d) => (d[0] + ", " + d[1]))
.attr("x", (d) => xScale(d[0] + 10))
.attr("y", (d) => yScale(d[1]));
```

Câu 1.13. {Bài tập – Thêm trục tọa độ}

```
const xAxis = d3.axisBottom(xScale);
const yAxis = d3.axisLeft(yScale);

svg.append("g")
.attr("transform", "translate(0," + (h - padding) + ")")
.call(xAxis);

svg.append("g")
.attr("transform", "translate(" + padding + ",0)")
.call(yAxis);
```

Câu 1.14. {Bài tập – Tổng kết}

```
<body>
	<script>
		const dataset = [
		[ 34, 78 ],
		[ 109, 280 ],
		[ 310, 120 ],
		[ 79, 411 ],
		[ 420, 220 ],
		[ 233, 145 ],
		[ 333, 96 ],
		[ 222, 333 ],
		[ 78, 320 ],
		[ 21, 123 ]
		];

		const w = 500;
		const h = 500;
		const padding = 60;

		const xScale = d3.scaleLinear()
		.domain([0, d3.max(dataset, (d) => d[0])])
		.range([padding, w - padding]);

		const yScale = d3.scaleLinear()
		.domain([0, d3.max(dataset, (d) => d[1])])
		.range([h - padding, padding]);

		const svg = d3.select("body")
```

```
                .append("svg")
                .attr("width", w)
                .attr("height", h);

                svg.selectAll("circle")
                .data(dataset)
                .enter()
                .append("circle")
                .attr("cx", (d) => xScale(d[0]))
                .attr("cy",(d) => yScale(d[1]))
                .attr("r", (d) => 5);

                svg.selectAll("text")
                .data(dataset)
                .enter()
                .append("text")
                .text((d) => (d[0] + "," + d[1]))
                .attr("x", (d) => xScale(d[0] + 10))
                .attr("y", (d) => yScale(d[1]))

                const xAxis = d3.axisBottom(xScale);
                const yAxis = d3.axisLeft(yScale);

                svg.append("g")
                .attr("transform", "translate(0," + (h - padding) + ")")
                .call(xAxis);

                svg.append("g")
                .attr("transform", "translate(" + padding + ",0)")
                .call(yAxis);
        </script>
    </body>
```

2. JSON APIs And Ajax

Câu 2.1. {Khái niệm}

- APIs là công cụ để các máy tính liên lạc với nhau, tức truyền và nhận dữ liệu. Ta thường sử dụng các kỹ thuật AJAX khi làm việc với APIs.
- AJAX = Asynchronous JavaScript And XML, là một nhóm các kỹ thuật thực hiện chức năng gửi các yêu cầu chuyển dịch dữ liệu đến server và load dữ liệu nhận được vào trang.
- Dữ liệu được chuyển dịch giữa trình duyệt và server thường ở dạng JavaScript Object Notation (JSON).

Câu 2.2. {Handle Click Events}

```
    // Đảm bảo code được thực thi sau khi trang loading.
    document.addEventListener('DOMContentLoaded',function(){
        document.getElementById('getMessage').onclick=function(){
            // Thay đổi dội dung khi click.
            document.getElementsByClassName('message')[0].textContent="Here is the
        message";
        }
    });
```

Câu 2.3. {Bài tập – Thao tác với JSON}

```
    // Ta cập nhật nội dung HTML với dữ liệu lấy từ API thông qua kỹ thuật AJAX.
    // Dữ liệu được chuyển có định dạng JSON, có thể  được chuyển thành JS Object thông
qua hàm JSON.parse().
    document.addEventListener('DOMContentLoaded',function(){
        document.getElementById('getMessage').onclick=function(){
            req = new XMLHttpRequest();                    // Tạo đối tượng request
```

```
		req.open("GET",'/json/cats.json',true);		// Khởi tạo request
		req.send();					// Gửi request
		req.onload = function(){			// Event-handler
			json=JSON.parse(req.responseText);
			document.getElementsByClassName('message')[0].innerHTML=JSON.stri
		ngify(json);
		console.log(json[2].codeNames[1]);		// In thử JASON
		};

		// Chuyển dữ liệu JSON về HTML.
		var html = "";					// Khai báo biến html
		json.forEach(function(val) {
			var keys = Object.keys(val);
			html += "<div class = 'cat'>";
			keys.forEach(function(key) {
				html += "<strong>" + key + "</strong>: " + val[key] +
			"<br>";
			});
			html += "</div><br>";
		});
		document.getElementsByClassName('message')[0].innerHTML = html;
	};
});

// Nội dung: Hiển thị hình ảnh ứng với đường dẫn được cho bởi thuộc tính imageLink.
// Đáp án: Thay các lệnh
var keys = Object.keys(val);
keys.forEach(function(key) {
	html += "<strong>" + key + "</strong>: " + val[key] + "<br>";
});
// (cont.) thành các lệnh.
html += "<img src = '" + val.imageLink + "' " + "alt='" + val.altText + "'>";

// Nội dung: Sử dụng filter() để lọc hình muốn hiển thị.
// Đáp án: Thêm lệnh này trước khi gọi đến json.forEach()
json = json.filter(function(val) {
	return (val.id !== 1);
});
```

Câu 2.4. {Bài tập – Tọa độ GPS}

```
// Đáp án: Lấy kinh độ và vĩ độ người dùng.
if (navigator.geolocation) {
	navigator.geolocation.getCurrentPosition(function(position) {
		document.getElementById('data').innerHTML = "latitude: "+
		position.coords.latitude + "<br>longitude: " +
		position.coords.longitude;
	});
}
```

Câu 2.5. {Bài tập – Gửi dữ liệu ra bên ngoài}

```
Req = new XMLHttpRequest();
req.open("POST",url,true);
req.setRequestHeader('Content-Type','text/plain');
req.onreadystatechange = function(){
	if(req.readyState==4 && req.status==200){
		document.getElementsByClassName('message')[0].innerHTML=req.responseText
```

```
        }
};
req.send(userName);
```

PHẦN B. 1.5 ĐIỂM

I. Apis And Microservices Certification

1. Managing Packages with Npm

1.1. Giới thiệu

- npm là công cụ dòng lệnh được sử dụng để chia sẻ và quản lý các mô-đun (hay các package) JS được viết trong NodeJS.
- Khi bắt đầu một project, npm tạo ra một file package.jason. File này sẽ liệt kê các package phụ thuộc của project.
- npm lưu các packages trong thư mục nodemodules. Các packages này có thể được thiết lập theo 2 cách: Toàn cục hoặc cục bộ.
- Sử dụng Glitch để post đáp án.

1.2. File package.json

- File package.json đóng vai trò trung tâm trong các ứng dụng NodeJS hay npm Package.
- File này chứa một đối tượng JASON duy nhất, trong đó thông tin được lưu trữ dưới dạng các cặp "key": value.
- Trong đối tượng này, hai trường bắt buộc là name và version. Tuy nhiên, ta có thể cung cấp thêm các thông tin khác để phục vụ cho những người dùng sau này.

1. Trường author: Tác giả project
"author": "Pham Huu Bao Chung",

2. Trường description: Mô tả project
"description": "A project that does something awesome",

3. Trường keyword: Tương tự như hashtag
"keywords": ["freecodecamp"],

4. Trường license: Quy định những quyền cơ bản của người dùng
"license": "MIT",

5. Trường version: Phiên bản hiện thời của project
"version": "1.0.0",

6. Trường dependencies: Các packages mà project cần.
"dependencies": {
"package-name": "version",
"express": "4.14.0",
"moment": "2.14.0"
},

- Lưu ý:
  - Phiên bản của npm packages (VD: moment) thường tuân theo quy chuẩn SemVer. Cấu trúc: MAJOR.MINOR.PATCH.
  - Ký hiệu ~
    "some-package-name": "~1.3.8" allows updates to any 1.3.x version
  - Ký hiệu ^
    "some-package-name": "^1.3.8" allows updates to any 1.x.x version.
  - Đáp án: https://chung-phb.glitch.me

2. Basic Node and Express

2.1. Giới thiệu

- Node.js là một công cụ hỗ trợ lập trình back-end (server-side). Node.js bao gồm nhiều mô-đun dựng sẵn:
  - HTTP: Mô đun đóng vai trò server
  - File System: Mô đun đọc và quản lý file
  - Path: Mô đun làm việc với đường dẫn thư mục và tệp
  - Assertion Testing: Mô đun kiểm tra code với các ràng buộc cho trước

- Express (không nằm trong Node.js) cũng là một mô đun thường được sử dụng.
    - Chạy giữa server được tạo bởi Node.js và các trang của ứng dụng web.
    - Định tuyến người sử dụng đến trang tương ứng với thao tác của họ.

2.2. Bài tập

// Câu 2.2. Khai báo đối tượng Express (thường sử dụng listen(port) để báo server kiểm tra một port nào đấy.)

```
var express = require('express');
var app = express();
const bodyParser = require("body-parser");
```

// Nội dung: Hàm middleware thường nhận 3 tham số: Đối tượng yêu cầu, đối tượng phản hồi và hàm tiếp theo trong chuỗi yêu cầu - phản hồi.

// Câu 2.8. Đưa thông tin của các request ra console.

```
app.use(function(req, res, next) {
      console.log(req.method + " " + req.path + " - " + req.ip);
      next();
});
```

// Câu 2.12. Nhận thông tin từ client với HTML forms thông qua request POST. Sử dụng package body-parser để giải mã dữ liệu. bodyParser.urlencoded({extended: false}) là middleware được dùng để xử lý dữ liệu URL.

```
app.use(bodyParser.urlencoded({extended: false}));
```

// Câu 2.1. Làm quen với NodeJS

```
console.log("Hello World");
```

// Nội dung: Định tuyến là phản hồi một request từ client đến một điểm đầu cuối nào đấy (bao gồm một đường dẫn và một phương thức HTTP request như GET, POST). Ta định nghĩa một tuyến theo cú pháp: app.METHOD(PATH, HANDLER), với METHOD = get, post, v.v. PATH là đường dẫn trên server và HANDLER được gọi đến khi route trùng khớp.

// Câu 2.3. Gửi "Hello Express" đến các GET requests ứng với đường dẫn gốc.

```
app.get('/', function (req, res) {
  res.send('Hello Express');
});
```

// Câu 2.4. Gửi file HTML đến các GET requests ứng với đường dẫn gốc.

```
var absolutePath = __dirname + "/views/index.html";
app.get('/', function (req, res) {
  res.sendFile(absolutePath);
});
```

// Câu 2.5. Gửi các tài nguyên tĩnh của ứng dụng. (Sử dụng hàm middleware express.static() để gửi. Hàm middleware là các hàm được dùng để chặn các route handler để bổ sung thông tin cần thiết. Chúng cần được mounted bởi app.use(path, middlewareFunction). Nếu path không được chọn thì middleware sẽ được thực thi với mọi loại request.)

```
app.use(express.static(__dirname + "/public"));
```

```
// Câu 2.6. Gửi đối tượng JSON.

app.get('/json', function (req, res) {
  res.json({"message": "Hello json"});
});

// Câu 2.7. Sử dụng file .env để truyền biến môi trường cho ứng dụng.

app.get('/json', function (req, res) {
  if(process.env.MESSAGE_STYLE === "uppercase") {
    res.json({"message": "HELLO JSON"});
  } else {
    res.json({"message": "Hello json"});
  }
});

/** 7) Root-level Middleware - A logger */
//  place it before all the routes !

// Nội dung: Middleware có thể được áp dụng đối với một route xác định (thông qua hàm app.METHOD(path, middlewareFunction)) và có thể được nối với các handler trong định nghĩa route.

// Câu 2.9. Nối middleware với handler để trả về thời điểm request.

app.get('/now', function(req, res, next) {
  req.time = new Date().toString();
  next();
}, function(req, res) {
  res.json({"time": req.time})
});

// Nội dung: Để cho phép người dùng lấy thông tin từ ứng dụng, ta có thể sử dụng tham số route.
// Ví dụ: route path: '/user/:userId/book/:bookId'
//            actual request URL: '/user/546/book/6754'
//            req.params: {userId: '546', bookId: '6754'}

// Câu 2.10. Xây dựng một echo server định vị tại route GET /:word/echo. Trả về một đối tượng JSON với cấu trúc {echo: word}.

app.get('/:word/echo', function(req, res) {
  res.json({"echo": req.params.word});
});

// Nội dung: Ta cũng có thể sử dụng query string để lấy input từ người dùng (đầu vào dữ liệu mà người dùng cần truy vấn đến).
// Dữ liệu từ query string được đưa vào đối tượng req.query.
// Ví dụ: route path: route path: '/library'
//            actual_request_URL: '/library?userId=546&bookId=6754'
//            req.query: {userId: '546', bookId: '6754'}
```

```
// Câu 2.11. Xây dựng một API endpoint, đinh vi tại GET /name. Trả về một đối tượng JSON với cấu trúc { name: 'firstname lastname'}. Các tham số firstname và lastname được lấy từ một query string, chẳng hạn: ?first=firstname&last=lastname.

// Lưu ý: Ở đây, thay vì viết app.get(path, handler), ta viết app.route(path).get(handler)(.post(handler)) để có thể áp dụng nhiều handler lên cùng một route.

app.route('/name')
  .get(function(req, res) {
    res.json({"name": req.query.first + " " + req.query.last});
  })
  .post(function(req, res) {
    res.json({"name": req.query.first + " " + req.query.last});
  });

// Câu 2.13. Lấy dữ liệu từ POST request.

app.route('/name').post(function(req, res) {
  res.json({"name": req.body.first + " " + req.body.last});
});

//---------- DO NOT EDIT BELOW THIS LINE --------------------
module.exports = app;
```

3. MongoDB and Mongoose

3.1. Giới thiệu

- MongoDB là một CSDL phi quan hệ, nghĩa là, Mongo lưu trữ tất cả dữ liệu có liên quan vào một bản ghi thay vì lưu trữ trên nhiều bảng khác nhau như trong SQL.
- Mongoose.js là một mô-đun npm cho phép viết đối tượng trong Mongo như trong JS.
- mLab được sử dụng để host một cơ sở dữ liệu miễn phí.

3.2. Bài tập

```
// Câu 3.1. Thiết lập
const mongoose = require('mongoose');
mongoose.connect("mongodb://chungphb:c251096@ds113873.mlab.com:13873/chungphb");

// Câu 3.2. Tạo model.
const personSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  age: Number,
  favoriteFoods: [String]
});

const Person = mongoose.model('Person', personSchema);

// Câu 3.3. Tạo và lưu một bản ghi (Lưu ý: Hàm callback (err, data) được sử dụng để handle các ngoại lệ.)
var createAndSavePerson = function(done) {
  const chung = new Person({
    name: "Chung",
    age: 22,
    favoriteFoods: ['Egg', 'Nacho']
```

```
  });

  chung.save((err, data) => err? done(err): done(null, data));
};

// Câu 3.4. Tạo nhiều bản ghi.
var createManyPeople = function(arrayOfPeople, done) {
  Person.create(arrayOfPeople, (err, data) => err? done(err): done(null, data));
};

// Câu 3.5. Tìm kiếm trên cơ sở dữ liệu.
var findPeopleByName = function(personName, done) {
  Person.find({"name": personName}, (err, data) => err? done(err): done(null, data));
};

// Câu 3.6. Tìm kiếm phần tử duy nhất trên cơ sở dữ liệu (Hoạt động tương tự find(), tuy nhiên, chỉ trả về một phần tử).
var findOneByFood = function(food, done) {
  Person.findOne({"favoriteFoods": food}, (err, data) => err? done(err): done(null,
data));
};

// Câu 3.7. Tìm kiếm dựa trên ID (được MongoDB cung cấp tự động).
var findPersonById = function(personId, done) {
  Person.findById(personId, (err, data) => err? done(err): done(null, data));
};

// Câu 3.8. Mô phỏng việc update thông qua các thao tác tìm kiếm, chỉnh sửa và lưu trữ.
var findEditThenSave = function(personId, done) {
  var foodToAdd = 'hamburger';
  Person.findById(personId, (err, data) => {
    if(err) {
      done(err);
    } else {
      data.favoriteFoods.push(foodToAdd);
      data.save((err, data) => err? done(err): done(null, data));
    }
  });
};

// Câu 3.9. Update thông qua hàm findOneAndUpdate().

var findAndUpdate = function(personName, done) {
  var ageToSet = 20;
  Person.findOneAndUpdate(
    {"name": personName},
    {$set: {"age": ageToSet}},
    {new: true},
    (err, data) => err? done(err): done(null, data));
};

// Câu 3.10. Loại bỏ phần tử thông qua hàm findByIdAndRemove() hoặc findOneAndRemove().
var removeById = function(personId, done) {
```

```
  Person.findByIdAndRemove(personId, (err, data) => err? done(err): done(null,
data));
};

// Câu 3.11. Loại bỏ nhiều phần tử thông qua hàm Model.remove().
var removeManyPeople = function(done) {
  var nameToRemove = "Mary";
  Person.remove({"name": nameToRemove}, (err, data) => err? done(err): done(null,
data));
};

// Câu 3.12. Thực hiện dãy thao tác.
var queryChain = function(done) {
  var foodToSearch = "burrito";

  Person
    .find({"favoriteFoods": foodToSearch})
    .sort({"name": "asc"})
    .limit(2)
    .select("-age")
    .exec(
      (err, data) => err? done(err): done(null, data)
    )};

/** **Well Done !!**
/* You completed these challenges, let's go celebrate !
 */

/** # Further Readings... #
/*  ====================== */
// If you are eager to learn and want to go deeper, You may look at :
// * Indexes ( very important for query efficiency ),
// * Pre/Post hooks,
// * Validation,
// * Schema Virtuals and  Model, Static, and Instance methods,
// * and much more in the [mongoose docs](http://mongoosejs.com/docs/)


//----- **DO NOT EDIT BELOW THIS LINE** ----------------------------------

exports.PersonModel = Person;
exports.createAndSavePerson = createAndSavePerson;
exports.findPeopleByName = findPeopleByName;
exports.findOneByFood = findOneByFood;
exports.findPersonById = findPersonById;
exports.findEditThenSave = findEditThenSave;
exports.findAndUpdate = findAndUpdate;
exports.createManyPeople = createManyPeople;
exports.removeById = removeById;
exports.removeManyPeople = removeManyPeople;
exports.queryChain = queryChain;
```

II. Information Security and Quality Assurance

1. Information Security with HelmetJS

```
var express = require('express'); // Do Not Edit
```

```
var app = express();              // Do Not Edit
// Câu 1.12. Sử dụng kỹ thuật băm BCrypt.
var bcrypt = require('bcrypt');
/** 1) Install and require `helmet` */
// Câu 1.1. HelmetJS được sử dụng để đề phòng thông tin riêng tư vô tình bị truyền giữa server và client.
const helmet = require('helmet');

/** 2) Hide potentially dangerous information - `helmet.hidePoweredBy()` */
Câu 1.2. X-Powered-By header có mặt trong tất cả các request của Express. Loại bỏ X-Powered-By header để hacker không biết ứng dụng được xây dựng trên nền tảng Express.
app.use(helmet.hidePoweredBy({ setTo: 'PHP 4.2.0' }));

/** 3) Mitigate the risk of clickjacking - `helmet.frameguard()` */(click chuột)
Câu 1.3. Không cho phép hacker đánh lừa người dùng bằng cách đặt trang web vào <frame> hoặc <iframe>.
app.use(helmet.frameguard( {action: 'deny'}));


/** 4) Mitigate the risk of XSS - `helmet.xssFilter()` */
Câu 1.4. Kỹ thuật XSS thường được sử dụng để ăn cắp các thông tin nhạy cảm của người dùng như cookies hay passwords. Sử dụng bộ lọc XSS để đề phòng mối nguy hại này.
app.use(helmet.xssFilter());

/** 5) Avoid inferring the response MIME type - `helmet.noSniff()` */
Câu 1.5. Không cho phép trình duyệt thay đổi Content-Type headers được cung cấp.
app.use(helmet.noSniff())

/** 6) Prevent IE from opening *untrusted* HTML - `helmet.ieNoOpen()` */
Câu 1.6. Đề phòng Internet Explorer mở các đoạn HTML không đáng tin cậy.
app.use(helmet.ieNoOpen());

/**  7) Ask browsers to access your site via HTTPS only - `helmet.hsts()` */
Câu 1.7. Chỉ cho phép trình duyệt truy cập trang web thông qua HTTPS để đề phòng protocol downgrade attacks và cookie hijacking.
var ninetyDaysInMilliseconds = 90*24*60*60*1000;
app.use(helmet.hsts({maxAge: ninetyDaysInMilliseconds,force: true}));

/** 8) Disable DNS Prefetching - `helmet.dnsPrefetchControl()` */
Câu 1.8. Vô hiệu hóa DNS Prefetching (đánh đổi hiệu năng).
app.use(helmet.dnsPrefetchControl());

/** 9) Disable Client-Side Caching - `helmet.noCache()` */
Câu 1.9. Vô hiệu hóa việc sử dụng bộ nhớ Cache (đánh đổi hiệu năng).
app.use(helmet.noCache());

/** 10) Content Security Policy - `helmet.contentSecurityPolicy()` */
Câu 1.10. Thiết lập và cấu hình Content Security Policy.
```

```
app.use(helmet.contentSecurityPolicy({
  directives: {
    defaultSrc: ["'self'"],
    scriptSrc : ["'self'", 'trusted-cdn.com']
  }
}));
```

Câu 1.11. Kết nối, cấu hình hoặc ngắt kết nối với các middleware thông qua configuration object.

```
app.use(helmet({
      frameguard: {                         // configure
            action: 'deny'
      },
      contentSecurityPolicy: {              // enable and configure
            directives: {
                  defaultSrc: ["self"],
                  styleSrc: ['style.com'],
            }
      },
      dnsPrefetchControl: false             // disable
      }));
```

Câu 1.12. Băm và so sánh mật khẩu một cách không đồng bộ.

```
bcrypt.hash(myPlaintextPassword, saltRounds, (err, hash) => {
  console.log(hash);
  bcrypt.compare(myPlaintextPassword, hash, (err, res) => {
      console.log(res);
  });
});
```

Câu 1.13. Băm và so sánh mật khẩu một cách đồng bộ.

```
var hash = bcrypt.hashSync(myPlaintextPassword, saltRounds);
var result = bcrypt.compareSync(myPlaintextPassword, hash);
console.log(result);

// ---- DO NOT EDIT BELOW THIS LINE --------------------------------------

module.exports = app;
var api = require('./server.js');
app.use(express.static('public'));
app.disable('strict-transport-security');
app.use('/_api', api);
app.get("/", function (request, response) {
  response.sendFile(__dirname + '/views/index.html');
});
var listener = app.listen(process.env.PORT || 3000, function () {
  console.log('Your app is listening on port ' + listener.address().port);
});
```

2. Advanced Node and Express

Câu 1.1. {Thiết lập PUG}

// PUG chứa các static template files hỗ trợ việc thiết kế các trang HTML và cho phép biểu diễn các biến lên trang mà không cần phải gọi đến API từ phía client.

```
app.set('view engine', 'pug');
```

```
app.route('/')
  .get((req, res) => {
    res.sendFile(process.cwd() + '/views/pug/index.pug');
  });
```

Câu 1.2. {Truyền biến từ server vào template file trước khi rendering vào HTML}

```
app.route('/')
  .get((req, res) => {
    res.render(process.cwd() + '/views/pug/index', {title: 'Hello', message: 'Please
login'});
  });
```

Câu 1.3. {Thiết lập Passport cho phép người dùng đăng ký hoặc đăng nhập tài khoản}

```
const session = require('express-session');
const passport = require('passport');

app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: true,
  saveUninitialized: true,
}));

app.use(passport.initialize());
app.use(passport.session());
```

Câu 1.4 – 1.5. {Serialization}

```
// Chuyển nội dung user object thành khóa và ngược lại.
mongo.connect(process.env.DATABASE, (err, db) => {
      if(err) {console.log('Database error: ' + err);
      else {
            console.log('Successful database connection');
            //serialization and app.listen
            passport.serializeUser((user, done) => {
                  done(null, user._id);
            });
            passport.deserializeUser( (id, done) => {
                  db.collection('users').findOne(
                        {_id: new ObjectID(id)},
                        (err, doc) => {
                              done(null, doc);
                        }
                  );
            });
      }
});
```

Câu 1.6. {Các chiến lược xác thực người dùng}

```
passport.use(new LocalStrategy(
      function(username, password, done) {
            db.collection('users').findOne({ username: username },
            function (err, user) {
                  console.log('User '+ username +' attempted to log in.');
                  if (err) { return done(err); }
                  if (!user) { return done(null, false); }
```

```
                        if (!bcrypt.compareSync(password, user.password)) { return
                        done(null, false); }
                        return done(null, user);
                  });
            }
      ));
      app.set('views', './views/pug' );
      app.route('/')
            .get((req, res) => {
                  res.render('index', {title: 'Home page', message: 'Please
                  login',showLogin: true, showRegistration: true});
             });
      function ensureAuthenticated(req, res, next) {
             if (req.isAuthenticated()) {
                  return next();
             }
             res.redirect('/');
      };
```

Câu 1.7. {Sử dụng passport strategies}

```
      app.route('/login')
            .post(passport.authenticate('local', { failureRedirect: '/' }),(req,res) => {
                  res.redirect('/profile');
            });
      app.route('/profile')
            .get(ensureAuthenticated, (req,res) => {
                  res.render('profile', {username: req.user.username});
            });
```

Câu 1.8. {Đăng ký người dùng mới}

```
      app.route('/register')
            .post((req, res, next) => {
                  db.collection('users').findOne({ username: req.body.username }, function
                  (err, user) {
                        if(err) {
                              next(err);
                        } else if (user) {
                              res.redirect('/');
                        } else {
                              var hash = bcrypt.hashSync(req.body.password, 12);
                              db.collection('users').insertOne(
                                    {username: req.body.username, password: hash},
                                    (err, doc) => {
                                          if(err) {
                                                res.redirect('/');
                                          } else {
                                                next(null, user);
                                          }
                                    }
                              );
```

```
                        }
                    })},
                    passport.authenticate('local', { failureRedirect: '/' }),
                    (req, res, next) => {
                          res.redirect('/profile');
                    }
                );
```

Câu 1.9. {Đăng xuất}

```
app.route('/logout')
      .get((req, res) => {
            req.logout();
            res.redirect('/');
      });

app.use((req, res, next) => {
      res.status(404)
            .type('text')
            .send('Not Found');
      });
```

Câu 1.10. {Code lẻ tẻ khác}

```
// Implementation of Social Authentication
app.route('/login')
  .post(passport.authenticate('local', { failureRedirect: '/' }), (req,res) => {
    res.redirect('/profile');
  });

// Implementation of Social Authentication II
passport.use(new GitHubStrategy({
    clientID: process.env.GITHUB_CLIENT_ID,
    clientSecret: process.env.GITHUB_CLIENT_SECRET,
    callbackURL: /*INSERT CALLBACK URL ENTERED INTO GITHUB HERE*/
  },
  function(accessToken, refreshToken, profile, cb) {
      console.log(profile);
      //Database logic here with callback containing our user object
  }
));

// Implementation of Social Authentication III
db.collection('socialusers').findAndModify(
    {id: profile.id},
    {},
    {$setOnInsert:{
        id: profile.id,
        name: profile.displayName || 'John Doe',
        photo: profile.photos[0].value || '',
        email: profile.emails[0].value || 'No public email',
        created_on: new Date(),
        provider: profile.provider || ''
    },$set:{
        last_login: new Date()
    },$inc:{
        login_count: 1
    }},
```

```
    {upsert:true, new: true},
    (err, doc) => {
        return cb(null, doc.value);
    }
);

// Set up the Environment
io.on('connection', socket => {
    console.log('A user has connected');
});
/*global io*/
var socket = io();


// Communicate by Emitting
socket.on('user count', function(data){
    console.log(data);
});

// Handle a Disconnect
socket.on('disconnect', () => { /*anything you want to do on disconnect*/ });

// Authentication with Socket.IO
io.use(passportSocketIo.authorize({
  cookieParser: cookieParser,
  key:          'express.sid',
  secret:       process.env.SESSION_SECRET,
  store:        sessionStore
}));

// Announce New Users
socket.on('user', function(data){
  $('#num-users').text(data.currentUsers+' users online');
  var message = data.name;
  if(data.connected) {
    message += ' has joined the chat.';
  } else {
    message += ' has left the chat.';
  }
  $('#messages').append($('<li>').html('<b>'+ message +'<\/b>'));
});
```