JAVASCRIPT

Dương Hữu Thành,
thanh.dh@ou.edu.vn,
Faculty of Information Technology,
Ho Chi Minh City Open University.

Agenda

- Introduction to Javascript
- Variables and Data types
- Introduction to ES6
- Javascript Scopes
- HTML DOM Interaction
- Function
- Object
- JS Prototype
- JS Class
- Call, Apply and Bind methods
- Module Pattern
- Synchronous and Asynchronous
- Promise and Async/Await

Introduction to JS

- Javascript (JS) is a client-side and interpreted language which supports object-oriented programming.
- JS enables to
  - Change the content of HTML elements.
  - Change the attributes of HTML elements.
  - Change CSS of HTML elements.
  - Validate user input at client side.

Data Types

- JS provides three primitive data types
  - Number
  - String
  - Boolean
- All numbers in JS are floating-point number, using 64 bits form of IEEE 754 standard.

Variables

- JS uses the var keyword to declare an variable and doesn't need to indicate the data types.
- JS variables can contain any certain values (untyped language).
- For example:

```
var x = 5;
x = true;
x = "good";
```

Introduction to ES6

- ECMAScript6 (ES6) is an advanced technique of programming in JS, source codes which are written by ES6 standard are cross-browser.
- The program in ES6 is cleaner, easy to maintain.

Template literals

- Template literals: this helps concat string more easily. Using pairs of `` and passing the arguments for the parameters in ${}.

```
function show(firstName, lastName) {
    return `Hello, ${firstName} ${lastName}`;
}

console.log(show("Thanh", "Duong"));
// Hello, Thanh Duong
```

let variable

- From ES6, var keyword is seldomly used because the variable is valid in any scopes and also outsite of block {}

```
function test (bool) {
    var amount = 0
    if(bool){
        var amount = 1;
    }
    return amount
}
console.log(test(true));
// 1
```

let variable

- Let variable is valid in block wrapping in {}.

```
function test (bool) {
    let amount = 0
    if(bool) {
        let amount = 1;
    }
    return amount
}
```

- Const variable helps its values are immutable.

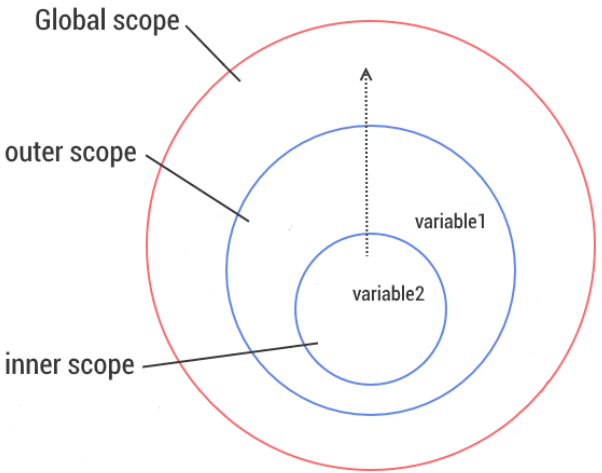
```
const a = 10;
```

Javascript Scopes

- Scope indicates the area where is valid to access variables or functions.
  - Global scope: variables or functions are declared in the global namespace and can access everywhere.
  - Function scope (local scope): variables or functions are declared within a function and can only access in the function.
  - Block scope (let, const): variables are declared within a block wrapping in {} and can only access within it.

Javascript Scopes

- The rules of finding variables have based the scopes. Firstly, JS finds in order from the current scope to outer scopes and global scope.



String

- In JS, String places between ‘’ or “”.
- Get the length of the String s: s.legnth
- Some methods of String
  - charAt(idx)
  - toLowerCase()/toUpperCase()
  - concat()
  - indexOf(s1 [, fromIndex])
  - lastIndexOf(s1 [, endIndex])

String

- Some methods of String
  - substring(fromIndex, endIndex)
  - substr(fromIndex, len)
  - slice(fromIndex, endIndex)
  - split(seperator)
  - replace(oldStr, newStr)
  - search(re)

Number

- Converting Number to String
  - toString()
  - toFixed()
  - toPrecision()
- Converting String to Number
  - Number()
  - parseFloat()
  - parseInt()

Array

- In JS, an array can contain the elements which are various data types.
  - Declare an array: a = []
  - Access the i-th element: a[i]
  - Get the number of elements: a.length

Array

- Some methods of Array
  - push()/unshift()
  - pop()/unshift()
  - indexOf(x [, fromIndex])
  - lastIndexOf(x, [, endIndex])
  - reverse()
  - sort()
  - forEach(function(value, index) {})
  - slice(beginIndex, endIndex)
  - concat()
  - join(seperator)

Date

- The Date object is used to manipulate with data of date and time (year, month, day, hour, minute, second, mili second).
- There are the various ways to create date
  - new Date() → return the current date
  - new Date(miliseconds)
  - new Date(dateString)
  - new Date(year, month, day, hours, minutes, seconds, miliseconds)

Date

- For example

```
// YYYY-MM-DD
var d1 = new Date("2017-05-27");
// YYYY-MM
var d2 = new Date("2017-05");
// YYYY
var d3 = new Date("2017");
// YYYY-MM-DDTHH:MM:SSZ
var d4 = new Date("2017-05-27T10:30:05Z");
```

Date

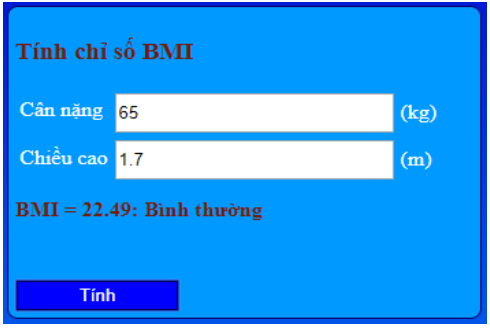
- Some methods
  - getDay() → get day of week (0-6)
  - getDate() → get day of month (1-31)
  - getMonth() → get month
  - getFullYear() → get four-digits year (yyyy)
  - getHours() → get hours
  - getMinutes() → get minutes
  - getSeconds() → get seconds
  - getMilliseconds() → get mili seconds

HTML DOM Interaction

- Some methods find the HTML elements.
  - document.getElementById()
  - document.getElementsByTagName()
  - document.getElementsByClassName()
  - document.getElementsByName()
  - document.querySelector()
  - document.querySelectorAll()

Assignment 1

- Write a program to calculate the BMI with the formular: BMI = weight (kg) / height^2 (m)



Below 18.5	Underweight
18.5 – 24.9	Normal or Healthy Weight
25.0 – 29.9	**Overweight**
30.0 and Above	**Obese**

Assignment 2

- Write a program to exchange between VNĐ to other currencies. Assume that:
  - 1 EUR = 26.000 VNĐ
  - 1 USD = 23.000 VNĐ
  - 1 AUD = 18.000 VNĐ

HTML DOM Interaction

- Changing the attributes of the HTML elements
  - Change text of an HTML element
    - element.innerHTML = <html>
    - element.innerText = <raw-text>
  - Change value of the attributes
    - element.<property> = <value
    - element.setAttribute(<attr-name>, <value>)

Assignment 3

ĐIỆN THOẠI SAMSUNG GALAXY S8+

Click to change the image

Samsung Galaxy S8+

20.050.000 VNĐ

Màu sắc

Click to change phone color

HTML DOM Interaction

- Changing the attributes of the HTML elements
  - Change css of HTML elements
    - element.style.<property> = <value>
  - Add events to HTML elements
    - element.<event> = function() {}
  - Listening an event
    - element.addEventListener(event, function, capture)

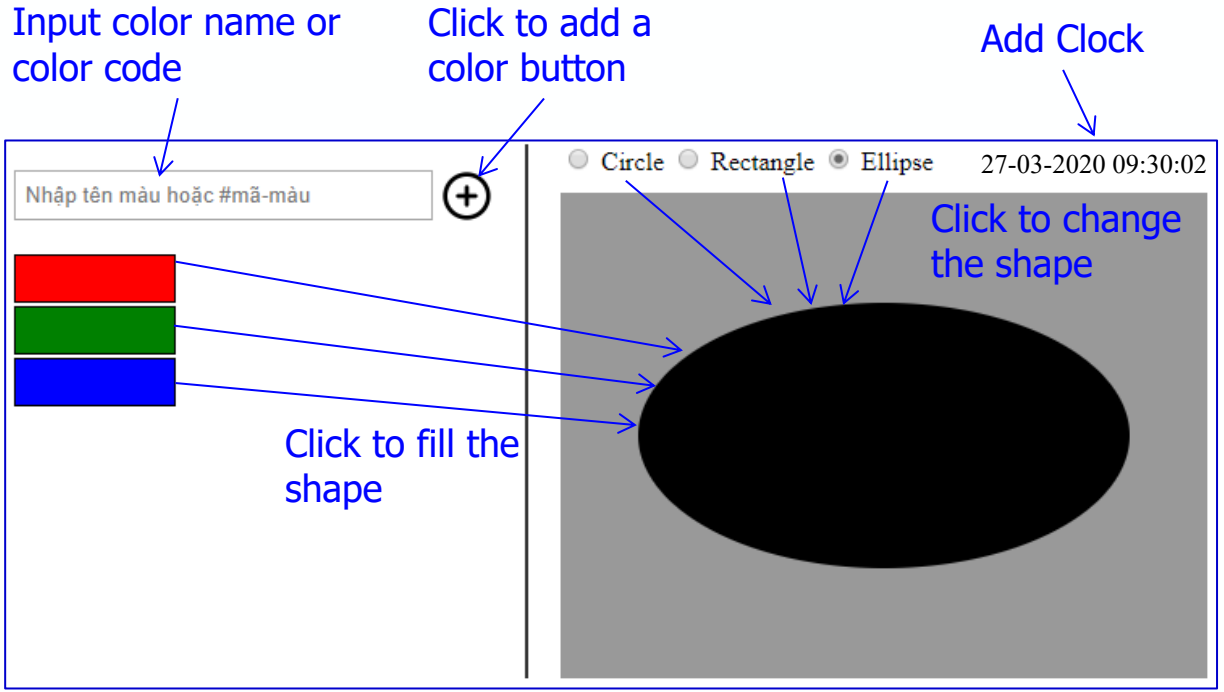
setInterval() and setTimeout()

- setTimeout(func, duration)
  - Invoking func after the duration.
- setInterval(<function>, <duration>)
  - Invoking func after every duration.
- clearTimeout(timeoutVariable)
- clearInterval(intervalVariable)

Assignment 4

- Write a program to display the current day and time with the following format:
  - Today is Friday, 03/07/2020
  - The current time: 03:57 PM
- Write a JavaScript program to rotate the string "Javascript" in right direction by periodically removing one letter from the end of the string and attaching it to the front.
  - Javascript → avascriptJ → vascriptJa → ...

Assignment 5



Function

- JS uses the function keyword to define a function which has zero or many parameters.

```
function funcName([Parameters]) {
    [statements]
    [return statement]
}
```

- Using the function by calling via function name and passing the arguments respectively.

Function

- If a parameter isn't passed a respective argument, it will be the undefined value.
- Moreover, the parameters of a function can delare the default value.

```
function add(a, b = 5) {
    return a + b;
}

console.log(add(5)); // 10
console.log(add(5, 7); // 12
```

Literal Function

- Literal function is introduced from JS 1.2

```
var varName = function([Parameters]) {
    [statements]
    [return statement]
}
```

- For example

```
var sum = function(a, b) {
    return a + b;
}
console.info(sum(2, 3));
```

Javascript Closures

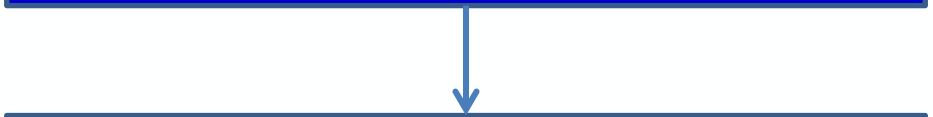
- Javascript supports nested functions which have access to the scope their parents' functions.
- A closure is an inner function that can access to the outer functions' variables.

```
function User(name){
    var displayName = function(greeting){
        console.log(greeting+' '+name);
    }
    return displayName;
}

var myFunc = User('Harvey');
myFunc('Welcome '); //Output: Welcome Harvey
```

Arrow Function

- This is a new way to define a function in JS.
- This one makes the functions cleaner because it ignores some unnecessary declarations.

```
function add(a, b = 5) {
    return a + b;
}
```

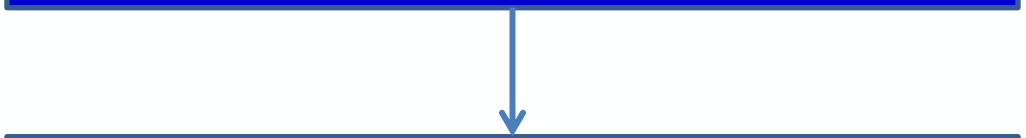


```
var add = (a, b = 5) => a + b;
```

Arrow Function

- For example

```
var a = [2, -5, 6, 9];
a.map(function(value) {
    return value + 1;
});
```



```
var a = [2, -5, 6, 9];
a.map((value) => value + 1);
```

Arrow Function

- For example

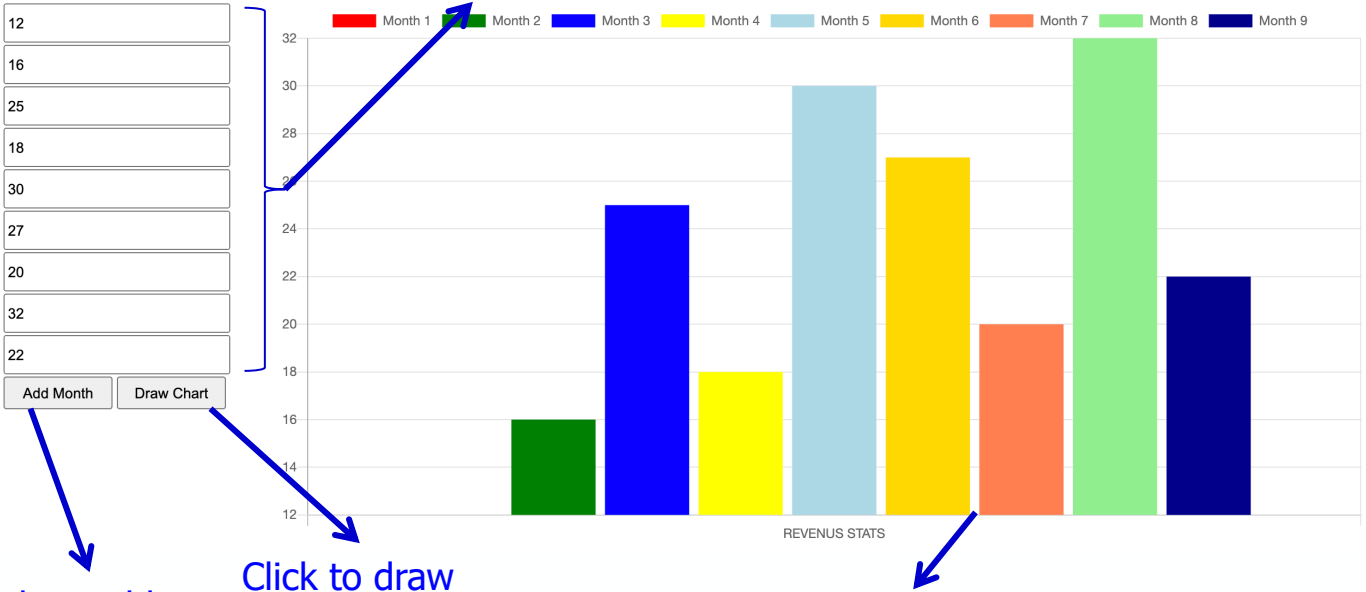
```
let a = [5, 8, 5, 9, -9, 4];

// Tăng mỗi phần tử của mảng lên 2 đơn vị
console.info(a.map(value =>  value + 2));

// Lấy các phần tử lẻ
console.info(a.filter(value => value % 2 != 0))

// Sắp xếp giảm
a.sort((t1, t2) => t2 - t1);
console.info(a);
```

Assignment 6

- Write a program to filter out the non-unique values of an array.

Assignment 7

Entering the month revenue, initialization has three textbox for three months



Click to add month having revenue, maximum = 12

Click to draw the chart based on the revenue data

Using chart.js for drawning the chart.
https://www.chartjs.org/docs/latest/
https://cdnjs.com/libraries/Chart.js

Callback Function

- A callback function is a function passed into another function as an argument.

```
var add = (a, b) => a + b;
var sub = (a, b) => a - b;

function execute(callback) {
    var a = 10, b = 20;
    return callback(a, b);
}

console.log(execute(add)); // 30
console.log(execute(sub));  // -10
```

Callback Function

- Callback function often used to continue some execution after an asynchronous operation has completed (calling asychronous callbacks).
- For example from jQuery syntax

```
$("#ball").show(3000, function() {
    $("body").css("background-color", "red");
});
```

Assignment 8

- Performing the CRUD functions of the categories by creating a mock API from https://mockapi.io/
  - Get categories to show on the web page.
  - Add/update a category.
  - Delete a category.
- In this assignment can use this mockapi
  - https://5efaff6280d8170016f75d51.mockapi.io/api/categories
- Notes: using jQuery to call API.

Multiple Expression

- It can evaluate multiple expressions in the same line, execute from the left to right and return the value of the last item on the most right side.
- For example

```
let x = 5;
function addFive(num) {
  return num + 5;
}
x = (x++ , x = addFive(x), x *= 2);

console.log(x) // 22
```

Object

- JS supports object-oriented programming. The built-in objects or user-created objects is derived from Object type.

```
var student = {
    firstName: "Le",
    lastName: "Duong",
    fullName: function() {
        return this.firstName + this.lastName;
    }
}
```

Object

- The this keyword refers to the value of the current object which executes or invokes the function.
- Arrow function do not bind their own this, instead, they inherit the one from the parent scope, which called “lexical scoping”.

Object

- Using the new operator to create the instance of an Object.

```
var user = new Object();
user.firstName = "Thanh";
user.lastName = "Duong";
```

Object

- For example

```
function user(name) {
    this.name = name;
    this.hello = function() {
        return "Hello " + this.name;
    }
}
var u = new user("Thanh");
console.info(u.hello());
```

Object

- Access the attributes and methods of an object

```
<object>.<attr-name>
```

```
<object>["<attr-name>"]
```

```
<object>.<method-name>([<arguments>])
```

Object

- JSON (Javascript Object Notation) is used widely to exchange data to/from web server.
- The methods
  - The JSON.parse(jsonText) method converts the json-like text to JS Object.
  - The JSON.stringify(jsObject) method converts JS Object to the json-like text.

Assignment 9

- Write a function to get the number of occurrences of each letters in specified string.

Assignment 10

- Write a program to read a json file as list of users in the following format:

```
[{
    "id": 1,
    "name": "Nguyen Van A",
    "dob": "1999-02-10"
}, {
    "id": 2,
    "name": "Tran Van C",
    "dob": "2000-09-15"
}]
```

  - Write a function to filter with the specified criteria (id, name or year of birth) by the passed argument.

Assignment 11

- Write a program to perform some tasks:
  - Read /api/dicts as dictionry words and display on the Web page.
  - Read a json file containing new words and add them to the dictionary via an mockapi /api/dicts. Notes if that is an existing word, displaying it on console.

JS Prototype

- JS prototype allows to add new properties and methods to object constructors.
- All JS objects inherit properties and methods from a prototype.
- An object which is created in JS is added a __proto__ property by JS engine. __proto__ points to the prototype object of the constructor function.

JS Prototype

- For example

```
function Student(name, yearOfBirth) {
    this.name = name;
    this.year = yearOfBirth;
}
Student.prototype.getAge = function() {
    var d = new Date();
    return d.getFullYear() - this.year;
}
var s = new Student("Nguyen Van A", 1995);
console.log(s.getAge());
```

JS Prototype

- For example: add the isDigitArray() method to Array objects.

```
Array.prototype.isDigitArray = function() {
    var isDigit = true;
    for (var i = 0; i < this.length && isDigit; i++)
        if (typeof this[i] != "number")
            isDigit = false;

    return isDigit;
};
var a = [5, 6, 3, 8, 5];
console.info(a.length); // 5
console.info(a.isDigitArray()); // true
```

Assignment 12

- Add the isLeap() method into Date object to check whether that the year of a date is leap year.
- Add the isPrime() method into Number object to check whether a number is prime.
- Add the countWords() method into String to count the occurencies of each words in a string.

JS Class

- JS class is introduced in ES6, it is a type of function, uses class keyword instead of function keyword to initiate the function.
- Using class always add the constructor() method which will be called each initialized the class object. Besides, we also add the own methods.

JS Class

```
class Point2D {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    calLen(p) {
        return Math.sqrt(Math.pow(this.x - p.x, 2)
                + Math.pow(this.y - p.y, 2));
    }
}
let p = new Point2D(2, 3);
let q = new Point2D(3, 4);
console.info(p.calLen(q)); // 1.4142135623730951
```

JS Class

- Class also allows to add getter and setter methods for the properties.
- Using static keyword to define static methods which are defined on the class itself, not on the prototype. So, order to use static method using class name.

```
class People {
    constructor(name) {
        this.name = name;
    }
    get name() {
        return this.name;
    }
    set name(n) {
        this.name = n;
    }
    static show() {

    }
}
```

JS Class

- Using extends keyword to execute a class inheritance, a sub class will inherit all the methods from the super class.

```
class People {
    constructor(name) {
        this.name = name;
    }
    show() {
        console.info(this.name);
    }
}
```

JS Class

```
class Lecturer extends People {
    constructor(name, degree) {
        super(name);
        this.degree = degree;
    }
    show() {
        super.show();
        console.info(this.degree);
    }
}

var lec = new Lecturer("Peter", "PhD");
lec.show();
```

Call, Apply and Bind methods

- The call() method allows a function/method belonging to an object to be assigned and called for a different object.
- The apply() method is used to pass arguments as an array or an array-like object.
- The bind() method returns a method instance instead of the result values. Later, it needs to execute a bound method with arguments.

Call, Apply and Bind methods

- Using call() to chain constructors for an object.

```
function People(name) {
    this.name = name;
}
function Lecturer(name, degree) {
    People.call(this, name);
    this.degree = degree;
}
var lec = new Lecturer("Nguyen Van A", "Dr");
console.info(`${lec.name} - ${lec.degree}`);
```

Call, Apply and Bind methods

- Using call() to invoke a function and specifying the context of this and the separated arguments

```
const student = {
    name: "Thanh"
}
function hello(city, country) {
    console.info("Hello %s, %s, %s.",
        this.name, city, country);
}
hello.call(student, "HCM City", "Vietnam");
```

Call, Apply and Bind methods

- Using apply() to invoke the function and the arguments are passed as array.

```
const student = {
   name: "Thanh"
}
function hello(city, country) {
   console.info("Hello %s, %s, %s.",
            this.name, city, country);
}
hello.apply(student, ["HCM City", "Vietnam"]);
```

Call, Apply and Bind methods

- Using the bind() method

```
const student = {
    name: "Thanh"
}
function hello(city, country) {
    console.info("Hello %s, %s, %s.",
        this.name, city, country);
}
var bound = hello.bind(student);
bound("HCM City", "Viet Nam");
```

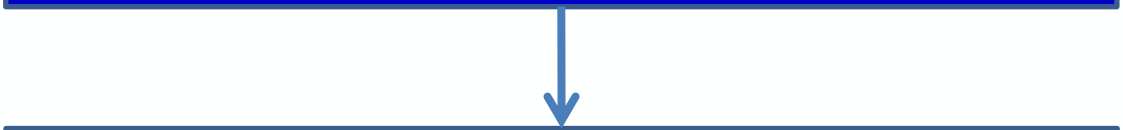
Module Pattern

- Module is just a script file. Modules can load and interchange functions each other.
- Module helps the program maintainablitity , resuability and namespacing.
- Using the export keyword marks variables or functions being able to access from outsite the current module.
- Using the import keyword is to import of functionality from other modules.

Module Pattern

- For example

```
// hello.js
export function hello(name) {
    console.info("Hello %s!!!", name);
}
```



```
// demo.js
import {hello} from "./hello.js";

hello("Thanh");
```

Synchronous and Asynchronous

- Synchronous: only executing next operation if the current operation has completed.
- Asynchronous: not waiting for each operation has completed, all operations will execute in the first. The result of each operation will be handled once it is available.

Promise

- Promise is a mechaism which allows to execute the asynchronous operations.

```
new Promise(function(resolve, reject) {
    ...
})
```

- The resolve function will be called when promise has completed.
- The reject function will be called when promise has failed

Promise

- For example

```
new Promise(resolve => resolve(1)).then((k) => {
    console.info(k);
    return k += 10;
}).then((k) => {
    console.info(k);
}).then(() => console.info("asychronous"));
console.log('synchronous');
```

```
synchronous
1
11
asychronous
```

Promise

- Some methods of promise object
  - then(onSuccess, onError): promise has completed successful.
    - Calling onSuccess if promise has completed
    - Calling onError if promise has occured error
  - catch(error): promise has rejected.

Async and Await

- In JavaScript asynchronous pattern handled in various versions,
  - ES5 → Callback
  - ES6 → Promise
  - ES7 → async & await
- The entire foundation for async/await is promise.
- Every async function returns promise, and every single thing awaits being ordinarily promise.

Async and Await

- The async keyword is used to declare a asynchronous function.
- The await keywords is used to pause the asynchronous function.

```
async function hello(name) {
    return "Hello " + name;
}
// true
console.info(hello("A") instanceof Promise);
// Hello A
hello("A").then((k) => console.info(k))
```


Q&A