# Javascript

## Kiểu dữ liệu

### Nguyên thủy :

|  |  |  |
| --- | --- | --- |
| **STT** | **Kiểu dữ liệu** | **Ví dụ** |
| **1** | number : kiểu số | 2 |
| **2** | string : kiểu chuỗi | “hiep.nh” |
| **3** | boolean : kiểu dữ liệu logic , đúng hoặc sai | true, false |
| **4** | undefine : Khi khai báo biến nhưng chưa gắn giá trị cho biến | undefine |
| **5** | null : Gắn giá trị cho biến thể hiện biến trống hoặc chưa tồn tại, không có giá trị | Null |
| **6** | symbol : Lưu trữ dữ liệu duy nhất và ẩn danh |  |
| **7** | bigInt : Kiểu số nhưng giá trị lớn hơn | 253 - 1 |

### Kiểu Object :

|  |  |  |
| --- | --- | --- |
| **STT** | **Kiểu dữ liệu** | **Ví dụ** |
| **1** | Object: kiểu đối tượng | const hiep = {  name :”Huy Hiep”;  age : 22;  log : function() {  return this.name + “ ” + this.age  }  } |
| **2** | Array: kiểu mảng | Const a =[12,8,”hiepnh”] |
| **3** | RegExp: Biểu thức chính quy |  |

## Array là gì ?

### Là một đối tượng chức các giá trị (thuộc mọi kiểu ) ở các vị trị được lập index

### Khởi tại 1 mảng

* const a =[]
* const b = new Array()

## Các phương thức trong Array

### push()

* Dùng để thêm phần tử cuối cùng của mảng
* Thay đổi mảng ban đầu
* Trả về độ dài của mảng

const plants = ['broccoli', 'cabbage', 'kale', 'tomato'];

console.log(plants.push("hihi")); // > 5

console.log(plants);

// > Array ["broccoli","cabbage", "kale","tomato","hihi"]

### pop()

* Dùng để xóa phần tử cuối cùng của mảng
* Thay đổi mảng ban đầu
* Trả về phần tử bị xóa, trả về undefined nếu mảng rỗng

const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop()); // > "tomato"

console.log(plants);

// > Array ["broccoli", "cauliflower", "cabbage", "kale"]

### shift()

* Dùng để xóa phần tử đầu tiên của mảng
* Thay đổi mảng ban đầu ,
* Trả về phần tử bị xóa , trả về undefined nếu mảng rỗng

const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.shift()); // > "broccoli"

console.log(plants);

// > Array ["cauliflower", "cabbage", "kale", "tomato"]

### unshift()

* Dùng để thêm phần tử vào vị trị đầu tiên của mảng
* Thay đổi mảng ban đầu
* Trả về độ dài của mảng

const plants = ['broccoli', 'cabbage', 'kale', 'tomato'];

console.log(plants.unshift("hihi")); // > 5

console.log(plants);

// > Array ["hihi""broccoli","cabbage", "kale","tomato"]

### concat()

* Dùng để nối 2 mảng với nhau\
* Không làm thay đổi mảng ban đầu và trả về một mảng mới

var array1 = ['a', 'b', 'c'];

var array2 = ['d', 'e', 'f'];

var array3 = array1.concat(array2);

console.log(array3); // > Array ["a", "b", "c", "d", "e", "f"]

console.log(array1); // > Array ["a", "b", "c"]

console.log(array2); // > Array ["d", "e", "f"]

### filter()

* Dùng để lọc ra các phần tử thỏa mãn điều kiện nào đó
* Không làm thay đổi mảng ban đầu và trả về một mảng mới
* Trả về một mảng rỗng nếu không có phần tử nào thỏa mãn

var words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

var result = words.filter(function(word) {

return word.length > 6;

});

console.log(result); // > Array ["exuberant", "destruction", "present"]

console.log(words); // > Array ["spray", "limit", "elite", "exuberant", "destruction", "present"]

### find()

* Dùng để lọc phẩn tử trong mảng, nó sẽ trả về phần tử đầu tiền tìm thấy trong mảng hoặc trả về undefined nếu không tìm thấy
* Không làm thay đổi mảng ban đầu

var array = [5, 12, 8, 130, 44];

var found = array.find(function(element) {

return element > 10;

});

console.log(found); // > 12

console.log(array); // > Array [5, 12, 8, 130, 44]

### foreach()

* Dùng để duyệt từng phần tử trong mảng
* Trả về undefined

var array = ['a', 'b', 'c'];

array.forEach(function(element) {

console.log(element);

});

// > "a"

// > "b"

// > "c"

console.log(array); // > Array ["a", "b", "c"]

### includes()

* Kiểm tra phần tử có tồn tại trong mảng , trả về true nếu tìm thấy , false khi không tìm thấy
* Không làm thay đổi mảng ban đầu

var array = [1, 2, 3];

console.log(array.includes(2)); // > true

console.log(array); // > Array [1, 2, 3]

### indexof()

* Trả về vị trí đầu tiên của phần tử , trả về -1 nếu không tìm thấy
* Không làm thay đổi mảng ban đầu

var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];

console.log(beasts.indexOf('bison')); // > 1

console.log(beasts.indexOf('bison', 2)); // > 4

console.log(beasts.indexOf('giraffe')); // > -1

console.log(beasts);

// > Array ["ant", "bison", "camel", "duck", "bison"]

### join()

* Để tạo ra một chuỗi mới bằng cách nối tất cả các phần tử của mảng , mặc định ngăn cách chúng với dấu phẩy hoặc 1 chuỗi ký tự xác định
* Không làm thay đổi mảng ban đầu , trả về chính phẩn tử đó nếu mảng có duy nhất một phẩn tử hoặc trả về “” nếu mảng rỗng

var elements = ['Fire', 'Air', 'Water'];

console.log(elements.join()); // > "Fire,Air,Water"

console.log(elements.join('')); // > "FireAirWater"

console.log(elements.join('-')); // > "Fire-Air-Water"

console.log(elements); // > Array ["Fire", "Air", "Water"]

### map()

* Dùng để tạo ra mảng mới với các phẩn tử là kết quả thực thi một hàm lên từng phần tử của mảng
* Không làm thay đổi mảng ban đầu

var array = [1, 4, 9, 16];

var map = array.map(function(item) {

return item \* item;

});

console.log(map); // > Array [1, 16, 81, 256]

console.log(array); // > Array [1, 4, 9, 16]

## Object là gì ?

### Đối tượng trong Javascript là một tập hợp các cặp khóa - giá trị, tương tự như bản đồ, từ điển, hay hash-table trong ngôn ngữ lập trình khác.

* Đối tượng là một tập hợp các thuộc tính
* Thuộc tính là một cặp khóa - giá trị chứa tên và giá trị
* Tên thuộc tính là một giá trị duy nhất có thể bị ép buộc vào một chuỗi và trỏ đến một giá trị
* Giá trị thuộc tính có thể là bất kỳ giá trị nào, bao gồm các đối tượng khác hoặc các hàm, được liên kết với tên/khóa

### Khởi tạo

* Dưới đây là ví dụ về tạo ra đối tượng rỗng trong Js:

// literal

const dog = { }

// constructor

const cat = new Object();

// static method

const horse = Object.create({ })

## Các vòng lặp trong Object

*const* hiepnh ={

    '*name*' : "*Huy Hiep*",

    "*age*" : 22,

    "*city*" :"*Ha Noi*"

  }

### for - in

*for* (*let* key *in* hiepnh) {

    console*.*log(key + " " + hiepnh[key]);

    }

### Object.keys

    Object*.*keys(hiepnh)*.*forEach(*function* (property) {

        console*.*log(key + " " + hiepnh[key]);

    })

### Object.getOwnPropertyNames()

    Object*.*getOwnPropertyNames(hiepnh)*.*forEach(*function* (key) {

    console*.*log(key + " " + hiepnh[key]);

    });

### Object.entries()

    Object*.*entries(hiepnh)*.*map((item) *=>* {

    console*.*log(item);

    });

    Object*.*entries(hiepnh)*.*forEach((item) *=>* {

    console*.*log(item);

    });

*for* (*const* item *of* Object*.*entries(hiepnh)) {

    console*.*log(item);

    }

## Function

### Function Declaration

* Function được khởi tạo bằng function tiếp theo là tên
* Sẽ được hoisting

*function* average(a, b, fn) {

*const* total = fn(a, b);

*return* total / 2;

  }

*let* result = average(200, 300, sum2);

### Function Expression

* Function được định nghĩa và lưu trữ trong một biến như một giá trị
* Function sẽ không được hoisting,

*const* logName = *function* () {

      console*.*log("*your name*");

    };

    logName();

### Anonymous function là gì ?

* Là 1 hàm ẩm danh , ko có tên gọi, sinh ra đúng vào thời điểm chạy của chương trình
* Xử lý quy mô nhỏ , tăng tốc độ và ko tốn nhiều bộ nhớ

  setTimeout(*function* () {

    console*.*log("*Được 1s rồi!*");

  }, 1000);

### IIFE là gì

* IIFE là viết tắt của Immediately Invoked Function Expression
* Là hàm đc thực thi ngay sau khi khởi tạo

  (*function* () {

    console*.*log("*this is IIFE function*");

  })();

### Arrow function là gì :

* Là 1 tính năng mới của es6
* Cú pháp ngắn gọn
* Hạn chế ảnh hưởng this trong 1 scope. Vì mỗi khi dùng function thì từ khóa this sinh ra

*const* plusNumber = (a, b) *=>* {

*return* a + b;

  };

### Currying function là gì trong JavaScript?

* Là biến 1 hàm n tham số thành n hàm 1 or ít đối số hơn

*function* multiply(a) {

*return* (b) *=>* {

*return* (c) *=>* {

*return* a \* b \* c

        }

    }

  }

  log(multiply(1)(2)(3))

### Closure trong javascript là gì?

* Là một chức năng có quyền truy cập vào phạm vi cha của nó , ngay cả khi scope đã đóng
* Là một hàm bên trong , truy cập giá trị ngoài phạm vi của nó

*function* sayHello2() {

*let* message = "*Hi FK*";

    // *inner function*

*function* sayHi() {

      console*.*log(message);

    }

*return* sayHi;

  }

*let* hello = sayHello2();

  hello();

### Callback function

* Là một hàm sẽ thực hiện sau khi một hàm khác được thực hiện xong
* Hàm là đối tượng , nên các hàm có thể lấy các hàm làm đối số và có thể trả về bởi một hàm khác

*function* add(a, b) {

*return* a + b;

  }

*function* average(a, b) {

*return* add(a, b) / 2;

  }

*let* result = average(10, 20);

  console*.*log(result);

## Câu điều kiện if else

*const* isRich = *true*;

*const* myMoney = 100;

*if* (!isRich) {

    // *!true -> false*

    console*.*log("*I will buy a new car*");

  } *else* *if* (myMoney *>* 1000) {

    console*.*log("*i will give you some money*");

  } *else* {

    console*.*log("*I poor*");

  }