



Functions

Ba Nguyễn

Functions

Functions (hàm - phương thức) được sử dụng để “đóng gói” các “khối mã” (**code block**), cho phép tái sử dụng code, giảm thiểu code, giảm lỗi, 🤔

💡 Tên hàm cũng sử dụng phong cách **camelCase**

Cú pháp khai báo:

```
// function declaration
function funcName() {
    // code
}
```

```
// run
funcName();
```

Biểu thức:

```
// function expression
let funcName = function () {
    // code
}
```

```
// run
funcName();
```

Functions

Ví dụ:

```
function isOdd() {  
    let n = +prompt("Nhập một số");  
    if (isNaN(n)) {  
        console.log("Vui lòng nhập một số hợp lệ!");  
    } else {  
        if (n % 2 == 1) {  
            console.log(n + " là số lẻ!");  
        } else {  
            console.log(n + " không phải là số lẻ!");  
        }  
    }  
}
```

// call function

`isOdd();`

`isOdd();`

`isOdd();`

`isOdd();`



Exercise



1. Viết hàm **maxOfThree()** cho phép nhập ba số, in ra số lớn nhất
2. Viết hàm **max()** cho phép nhập lượng số tùy ý in ra số lớn nhất
3. Viết hàm **isPrime()** cho phép nhập một số, kiểm tra và in ra số đó có phải số nguyên tố hay không

Local vs Global variables

Một biến được khai báo bên trong hàm **chỉ tồn tại bên trong hàm** đó, biến đó được gọi là **local variable**

```
function local() {  
    let name = "Ba";  
    console.log(name);  
}  
local(); // "Ba"  
console.log(name); // ❌ error
```

Một biến được khai báo bên ngoài tất cả các hàm (khối code) được gọi là **global variable**, hàm có thể truy cập tới biến khai báo bên ngoài

```
let name = "Ba";  
function global() {  
    console.log(name);  
}  
global(); // "Ba"  
console.log(name); // "Ba"
```


💡 Hàm chỉ tham chiếu tới biến bên ngoài, **nếu** trong hàm không có biến **local** nào trùng tên, ngoài ra, **không nên** tham chiếu trực tiếp tới biến **global**, thay vào đó, sử dụng **parameters**

Function Parameters

Tham số (đối số) cho phép truyền giá trị từ bên ngoài vào hàm, tham số được khai báo cùng với hàm. Khi một hàm được gọi, nó sẽ **sao chép** giá trị truyền vào và gán cho tham số tương ứng

```
function isOdd(number) {  
    // khai báo hàm với tham số number  
    if (number % 2 === 0) {  
        console.log(number + " không phải số lẻ");  
    } else {  
        console.log(number + " là số lẻ");  
    }  
}  
  
// gọi hàm  
isOdd(1); // number = 1  
isOdd(2); // number = 2
```

```
let outName = "Ba";  
  
function demo(inName) {  
    inName = "Béo Ú";  
    console.log(inName);  
}  
  
demo(outName); // "Béo Ú"  
console.log(outName); // "Ba"
```



Default Parameters

Khi hàm được gọi mà không truyền giá trị, tham số sẽ nhận giá trị **undefined**. Có chỉ định một giá trị mặc định cho tham số sẽ được sử dụng trong trường hợp đó:

```
function hi(name) {  
    console.log("Hello " + name);  
}
```

```
hi(); // "Hello undefined"  
hi("Ba"); // "Hello Ba"
```

```
function hi(name = "stranger") {  
    console.log("Hello " + name);  
}
```

```
hi(); // "Hello stranger"  
hi("Ba"); // "Hello Ba"
```



Return

Một hàm luôn trả về một giá trị nào đó khi nó được gọi, mặc định là **undefined**. Để chỉ định một giá trị trả về từ hàm, sử dụng câu lệnh **return**:

```
function hi(name) {  
    console.log(name);  
}
```

```
hi("Ba");  
// log "Ba"  
// then return undefined
```

```
function isOdd(number) {  
    return number % 2 === 1; // return result  
}
```

```
let check = isOdd(5); // true  
  
if (isOdd(5)) {  
    // code  
}
```



Câu lệnh **return** tương tự **break**, nó “dừng” hàm, bỏ qua mọi câu lệnh bên dưới

Exercise



1. Viết hàm **printPrime(n)** in ra các số nguyên tố trong khoảng từ $0 \rightarrow n$, mặc định $n = 100$
2. Viết hàm **convertTemperature(temp, to)** chuyển đổi và in ra nhiệt độ từ Celsius sang Fahrenheit hoặc Kevin, mặc định sẽ chuyển từ Celsius sang Kevin

Functions

💡 Một số lưu ý khi sử dụng hàm

- Một hàm chỉ nên thực hiện một công việc duy nhất, nếu một hàm phức tạp với nhiều công việc nhỏ, hãy tách ra thành các hàm riêng biệt
- Tên hàm nên mô tả chính xác chức năng của nó, ví dụ: `isOddNumber()`, `calcAverage()`, ...

💡 Lưu ý khi dùng `return` với toán tử ?

```
condition ? return value1 : return value2; // ❌ error  
return condition ? value : value; // ✅ oke
```

💡 Có thể sử dụng `return;` để ngắt hàm mà không trả về giá trị nào (**undefined**)

Homework

1. Viết hàm tính và trả về lập phương của một số n , n là tham số truyền vào
2. Viết hàm tính tổng dãy số Fibonacci từ $0 \rightarrow n$, n là tham số truyền vào.
3. Viết hàm kiểm tra n có phải số Prime, kết quả trả về true hoặc false, n là tham số truyền vào
4. Viết hàm kiểm tra n có phải số Armstrong, kết quả trả về true hoặc false, n là tham số truyền vào
5. Viết hàm kiểm tra n có phải số Perfect, kết quả trả về true hoặc false, n là tham số truyền vào
6. Viết hàm kiểm tra n có phải số Palindrome, kết quả trả về true hoặc false, n là tham số truyền vào
7. Viết hàm tính và trả về giai thừa của n , n là tham số truyền vào
8. Viết hàm kiểm tra số a có phải lũy thừa của b hay không, kết quả trả về true hoặc false, a và b là tham số truyền vào
9. Viết hàm tính và trả về hiệu giữa tích và tổng của các chữ số trong một số n .
10. Viết hàm tính và trả về kết lũy thừa bậc y của x , với x và y là tham số truyền vào (y có thể âm)

Arrow functions

Arrow function là cú pháp mới, cho phép viết hàm ngắn gọn hơn và bao gồm một số tính năng khác biệt so với hàm thông thường (sẽ được đề cập sau)

```
let yes = () => console.log("I love you too!");
```

```
let yes = function () {  
    console.log("I love you too!");  
};
```

```
let square = (n) => n * n;
```

```
let square = function (n) {  
    return n * n;  
};
```

Arrow functions

Arrow function là cú pháp mới, cho phép viết hàm ngắn gọn hơn và bao gồm một số tính năng khác biệt so với hàm thông thường (sẽ được đề cập sau)

```
let pow = function (x, y) => {  
  let result = x;  
  if (y > 0) {  
    for (let i = 1; i < y; i++) { result *= x; }  
  } else {  
    for (let i = 0; i ≥ y; i--) { result /= x; }  
  }  
  return result;  
};
```

Homework

1. Viết hàm `random(a, b)`, trả về số ngẫu nhiên trong khoảng $a - b$
2. Viết hàm `isTriangle(a, b, c)` nhận vào 3 tham số là cạnh của tam giác, kiểm tra 3 cạnh có phải tam giác hợp lệ hay không?
3. Viết hàm `guessNumber()`, khởi tạo một số ngẫu nhiên trong khoảng 0 - 10, sau đó hiển thị bảng cho phép người dùng đoán giá trị. Nếu đoán đúng hiển thị "Â mây zing, gút chóp" và dừng hàm, nếu sai hiển thị "Không khớp" và cho phép nhập lại. Số lần nhập tối đa là 3 lần, hoặc người dùng có thể bấm cancel để dừng hàm.
4. Viết hàm `nearest(a, b)` kiểm tra và trả về số gần với 100 nhất. VD: `nearest(90, 105) // 105`, hoặc `nearest(80, 90) // 90`
5. Viết hàm `sequence(a, b, c)` kiểm tra xem a, b, c có có phải tăng dần (hoặc giảm dần) hay không.

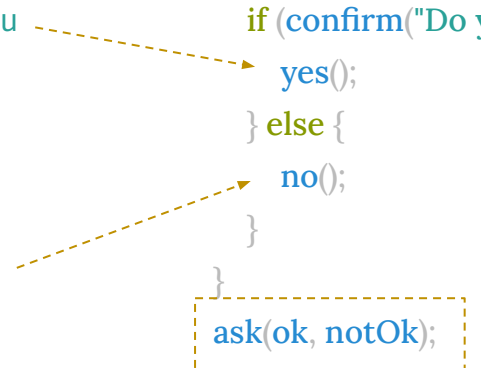
Homework

1. Viết hàm `countDecimal(number)` trả về số chữ số trong phần thập phân của `number`
2. Viết hàm `isAscending(number)` kiểm tra dãy số của `number` có phải dãy tăng dần hay không?. VD: `isAscending(123) // true`
3. Viết hàm `countEven(number)` trả về số chữ số chẵn của `number`. VD: `countEven(12345) // 2`
4. Viết hàm `find(number)` trả về số `n` sao cho $1 + 2 + \dots + n \leq \text{number}$. VD: `find(7) // 3`
5. Viết hàm `sum(value1, unit1, value2, unit2)` quy đổi về cùng đơn vị và trả về tổng giá trị `value1 + value2` theo `unit1`, `unit` bao gồm km, m, cm, dm, mm. VD: `sum(1, 'km', 100, 'm') // 1,1 km`

Callback

Hàm cũng chỉ là một giá trị, nó có thể được gán cho một biến, sao chép, ... Một hàm được truyền vào hàm khác dưới dạng tham số được gọi là hàm **callback**

```
function ok() {  
    console.log("I love you  
too!");  
}  
  
function notOk() {  
    console.log("👊 you");  
}  
  
function ask(yes, no) {  
    if (confirm("Do you love me!")) {  
        yes();  
    } else {  
        no();  
    }  
}  
  
ask(ok, notOk);
```



💡 Cần chú ý các cuộc gọi hàm lồng nhau (**callback hell**) VD: $a(b) \rightarrow b(a) \rightarrow a(b) \rightarrow \dots$

Scope

Scope chỉ phạm vi tồn tại của một biến. Một biến chỉ tồn tại bên trong khối (**block**) code mà nó được khai báo. Một **block** là đoạn code được đặt trong cặp dấu **{ }**. Các cấu trúc điều khiển, hàm, class là một block.

💡 Riêng biến khai báo với **var** chỉ bị giới hạn trong **block của hàm**

```
{  
  let x = 1;  
  const y = 2;  
  var z = 3;  
}  
  
alert(x); // ❌ error  
alert(y); // ❌ error  
alert(z); // 🔥 3
```

```
if (true) {  
  let x = 1;  
  const y = 2;  
  var z = 3;  
}  
  
alert(x); // ❌ error  
alert(y); // ❌ error  
alert(z); // 🔥 3
```

```
function a() {  
  let x = 1;  
  const y = 2;  
  var z = 3;  
}  
  
alert(x); // ❌ error  
alert(y); // ❌ error  
alert(z); // ❌ error
```

Scope

Ví dụ với vòng lặp:

```
for (let i = 0; i < 10; i++) {  
    // code  
}  
console.log(i); // ❌ error i is not defined
```

```
for (var i = 0; i < 10; i++) {  
    // code  
}  
console.log(i); // i = 10
```

Hoisting

Với cú pháp khai báo hàm thông thường, hàm có thể được gọi trước khi nó được khai báo, tuy nhiên, với biểu thức hàm, hàm sẽ chỉ gọi được sau khi nó được khai báo

```
hi(); // 🟡 oke  
function hi() {  
    // code  
}
```

```
bye(); // ❌ error  
let bye = function() {  
    // code  
}  
bye(); // 🟡 oke
```



💡 Khi thực thi, JavaScript “nhắc” (**hosting**) tất cả khai báo hàm (và cả biến nếu khai báo với **var**) lên đầu tập lệnh hoặc khối code

Hoisting

Với biến được khai báo bởi **var**, nó cũng được “nhắc” lên đầu giống như hàm, tuy nhiên, giá trị chỉ được gán cho tới đúng vị trí câu lệnh trong mã, **hosting** không hoạt động với **let** và **const**

```
alert(name); // undefined  
var name = "Ba";  
alert(name); // "Ba"
```

```
var name; // undefined  
alert(name); // undefined  
name = "Ba";  
alert(name); // "Ba"
```

```
alert(other); // error  
let other = "😞";  
alert(other); // "😞"
```



Closure

Trong JavaScript, hàm có thể ghi nhớ và truy xuất tới giá trị của các biến được khai báo bên ngoài nó, kết hợp với **scope**, chúng ta có thể “**ẩn**” các giá trị khỏi mã bên ngoài, nhưng cho phép truy cập thông qua một hàm, đó được gọi là **closure**

```
let counter = (function() => { // closure
    let count = 0; // biến cục bộ
    return () => count++; // trả về một hàm, hàm này có thể truy xuất tới biến count bên ngoài
})(); // hàm tự thực thi, giá trị của counter là hàm () => count++
```

```
console.log(counter); // () => count++
console.log(count); // ❌ error, mã bên ngoài không thể truy cập biến “count”
counter(); // return 0, count = 1
```





Objects

Ba Nguyễn

What is object?



Object (đối tượng) là kiểu dữ liệu đặc biệt, và cũng là khái niệm quan trọng nhất khi tìm hiểu về JavaScript, bởi hầu hết mọi thứ trong JavaScript đều liên quan tới object

Object mô phỏng một đối tượng thực tế trong ngôn ngữ lập trình, mỗi object bao gồm 2 thông tin mô tả về đối tượng đó: **Properties** (thuộc tính) và **methods** (phương thức)

Mỗi property mô tả một thông tin về đối tượng, nó bao gồm một cặp **key** (khóa) và **value** (giá trị). Một object tất nhiên có thể có nhiều properties. Giá trị của thuộc tính có thể là bất kỳ kiểu dữ liệu nào, đơn giản như string, number, hay phức tạp hơn như một hàm (method), hoặc một object con

Mọi thông tin của **object** đều được truy xuất thông qua **key**

Objects

Ví dụ:

- Mỗi **person** là một object, có các properties như **name**, **age**, **job**, ... và các methods như **speak**, **laugh**, **eat**, ...
- Mỗi **computer** là một object, có các properties như **brand**, **series**, **size**, **cpu**, **memory**, ..., và các methods như **start**, **shutdown**, ...
- Mỗi **cat** là một object, có các properties như **name**, **breed**, **weight**, ..., và các methods như **meow**, **run**, **bite**, ...
- Mỗi **character** là một object, có các properties như **name**, **level**, **weapon**, **damage**, ... và các methods như **attack**, **run**, ...



Objects

Cú pháp khai báo **object**:

```
// khai báo object rỗng  
let ba = {}; // literal
```

```
// khai báo kèm thuộc tính  
let ba = {  
  name: "Ba",  
  age: 28,  
  job: "Developer",  
  mobile: mobile,  
};
```

```
let mobile = new Object(); // constructor
```

```
// thêm thuộc tính  
mobile.brand = "iPhone";  
mobile.version = "100 Pro Maximum";  
mobile.phoneNumber = "0965338283";
```

```
// xóa thuộc tính  
delete ba.job;  
delete ba.name;  
delete ba.age;
```



Objects

Truy cập thông tin của object, sử dụng cú pháp **object.key** hoặc **object[key]**:

```
ba.name; // "Ba"  
ba["name"]; // "Ba"  
ba.mobile; // mobile {}  
ba.mobile.brand; // "iPhone"  
ba.mobile["version"];  
// "100 Pro Maximum"  
ba["mobile"].phoneNumber;  
// "0965338283"
```

```
ba.job = "Teacher"; // change value  
ba["mobile"]["version"] = "Xs"; // change  
value  
ba.mobile.brand = "iPorn"; // change value  
ba.name = "Ba đẹp trai 😎"; // change value  
ba.age = 82; // change value  
ba["age"] = 2020 - 1992; // change value
```



Objects

💡 Một số lưu ý về object

- **Key** được lưu với kiểu dữ liệu **string** (mọi kiểu dữ liệu khác đều được chuyển về kiểu string)
- **Key** không bị giới hạn về đặt tên giống như biến, nó có thể chứa ký tự đặc biệt, hoặc trùng với *keyword* trong Javascript
- Cú pháp truy cập properties: **object.key** hoặc **object[key]**
- Có thể thêm, thay đổi giá trị, hoặc xóa một properties khỏi object
- Khi truy cập một property không tồn tại trong object, giá trị trả về sẽ là **undefined**

💡 Cú pháp **object[key]** thường dùng với biến

```
let key = "name";  
obj[key]; // obj["name"]  
obj[key] = "Ba"; // obj["name"] = "Ba"
```

Objects

Khi sử dụng một biến làm **key**, đồng thời nhận chính giá trị của biến đó làm **value**, có thể sử dụng cú pháp viết tắt

```
let name = "Ba";  
let age = 28;  
let job = "Gamer";
```

```
let ba = {  
  name, // "name": name  
  age,  // "age": age  
  job,  // "job": job  
};
```

```
ba.name; // "Ba"
```

```
let key = prompt("Enter key");  
let value = prompt("Enter value");
```

```
let obj = {  
  [key]: value,  
};
```

```
// VD: key = abc, value = 123  
// [key]: value → abc: 123
```

Exercise



1. Viết hàm cho phép nhập key và value tương ứng, gán key và value vào một object. Hàm cho phép nhập số lượng key và value tùy ý, chỉ dừng khi người dùng nhập key rỗng hoặc null. Sau đó in object ra console

Objects

Kiểm tra một **key** có tồn tại trong **object** hay không sử dụng toán tử **in**

```
let user = {}; // empty
"name" in user; // false
user.name = "Ba";
"name" in user; // true
```

```
let key = "age";
key in user; // false
user[key] = 28;
key in user; // true
```

Lặp qua tất cả thuộc tính (*) trong **object**, sử dụng cú pháp **for in**

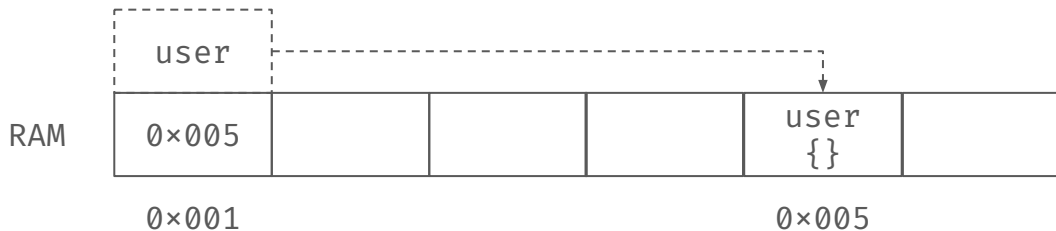
```
let user = {
  name: "Ba",
  age: 28,
  job: "Developer",
};
```

```
for (let k in user) {
  console.log(k + ": " + user[k]);
}
// name: Ba
// age: 28
// job: Developer
```

References

Một biến lưu trữ đối tượng không thực sự lưu trữ thông tin về đối tượng đó, mà lưu trữ địa chỉ vùng nhớ của đối tượng trong bộ nhớ máy tính (tham chiếu) (trong khi các kiểu dữ liệu cơ bản là kiểu tham trị - biến lưu trữ trực tiếp giá trị)

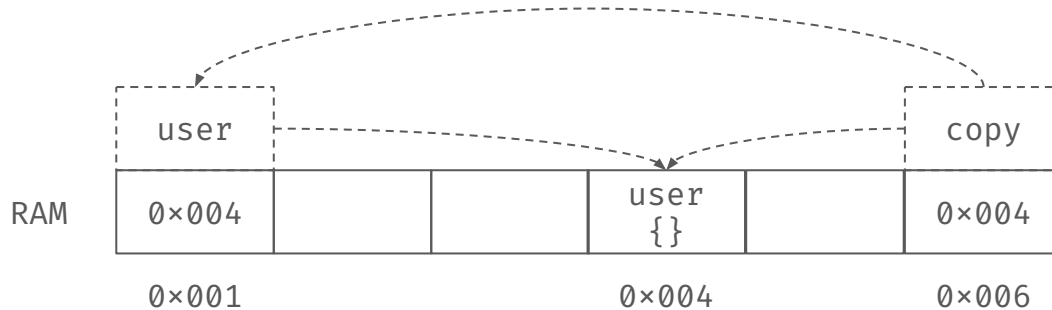
Khi truy xuất thông tin đối tượng, JavaScript di chuyển tới địa chỉ thực của đối tượng thông qua biến và thao tác trên đó



References

Khi sao chép một đối tượng cho một biến, nó sao chép địa chỉ vùng nhớ (sao chép tham chiếu)

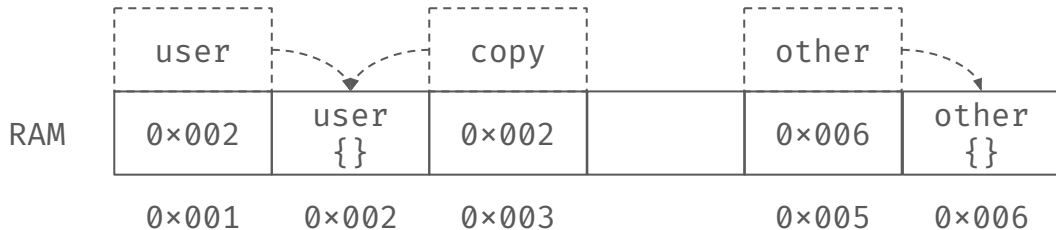
Khi 2 biến cùng tham chiếu tới một đối tượng, một biến thay đổi giá trị của đối tượng, biến còn lại cũng nhận được sự thay đổi đó



References

Khi so sánh 2 đối tượng, nó so sánh địa chỉ tham chiếu (không phải giá trị của đối tượng)

```
let user = {}; // empty
let copy = user;
user = copy; // true
let other = {}; // empty
user = other; // false
```



References

Ví dụ về tham chiếu trong trường hợp sử dụng hàm và sao chép biến

```
function demo(obj) {  
    obj.name = "Changed";  
}
```

```
let user = { name: "Ba" };
```

```
demo(user);
```

```
user.name; // "Changed"
```

```
let obj = {  
    name: "Ba",  
};
```

```
let copy = obj; // copy
```

```
copy.name = "Changed";
```

```
obj.name; // "Changed"
```

Exercise



1. Viết hàm **double(obj)** nhận vào tham số là một **object**, hàm cập nhật tất cả giá trị có kiểu **number** trong **obj** thành bình phương của chính giá trị đó. Gọi hàm và hiển thị **obj** ra console

Object copy

Để sao chép đối tượng (sao chép giá trị, không phải tham chiếu), sử dụng **for in** hoặc **Object.assign()**:

```
let obj = { name: "Ba", age: 28, ... };
```

```
for (let k in obj) {  
    copy[k] = obj[k];  
}
```

```
copy.name = "Changed";  
obj.name; // "Ba"
```

```
let copy = {};
```

```
Object.assign(copy, obj);
```

```
copy.age = 82;  
obj.age; // 28
```

? Nếu object chứa một object con thì sao???



Exercise



1. Viết hàm **copy(target, source)** sao chép toàn bộ thông tin của đối tượng **source** tới đối tượng **target**, sử dụng **for in**. Thử thay đổi một vài giá trị trong target hoặc source, sau đó in cả 2 object ra console

Object methods

💡 Mỗi **object** bao gồm 2 thông tin: thuộc tính (properties) và phương thức (methods), thuộc tính mô tả một thông tin về đối tượng, còn phương thức mô tả một hành động (chức năng) của đối tượng đó

Cú pháp khai báo phương thức cho **object**:

```
let user = {  
  a() { // code },  
  b: function () { // code },  
};
```

```
user.c = function () {  
  // code  
};
```

```
// call object method  
user.a();  
user.b();  
user.c();
```



this

Để có thể truy cập được tới các properties của object, các methods tồn tại một *toán tử - biến - từ khóa* **this** tham chiếu tới chính object gọi nó, thông qua **this**, methods có thể truy cập được tới các properties của object

```
let user = {  
  name: "Ba",  
  hi() {  
    console.log(this.name);  
  },  
};
```

```
user.hi(); // this = user
```

```
function hi() {  
  console.log(this.name);  
}
```

```
let user = {  
  name: "Ba",  
  hi,  
};
```

```
user.hi();
```



💡 Giá trị của **this** không được xác định khi khai báo methods, nó được xác định methods được gọi

Exercise

1. Tạo một object user chứa các thông tin name, age, ...
2. Thêm một phương thức hi() cho user, in ra câu chào dạng “Hello, my name is ...”
3. Thêm phương thức getKeys() in ra toàn bộ key của user, sử dụng for in
4. Thêm phương thức getValues() in ra toàn bộ value của user, sử dụng **for in**

💡 JavaScript cung cấp sẵn một số phương thức như `Object.keys(obj)`, `Object.values(obj)`, và `Object.entries(obj)` để lấy ra key và value của **obj**

this

Arrow function không tồn tại biến - từ khóa - toán tử **this**, nó tham chiếu tới **this** bên ngoài như một biến thông thường

```
let ba = {  
  name: "Ba",  
  hi: () => console.log(this.name),  
};
```

```
ba.hi(); // undefined
```

```
let beo = {  
  name: "Béo",  
  hi() { // có "this"  
    // arrow function  
    return () => console.log(this.name);  
  },  
};
```

```
beo.hi(); // "Béo"
```

? Giá trị của **this** khi một phương thức được gọi mà không phải bởi object??? **globalThis**

Constructor

Cú pháp **hàm khởi tạo** cho phép tạo ra nhiều đối tượng dựa trên một “bản mẫu” với danh sách properties và methods được xác định trước

```
function Person(name, age) {  
    this.name = name;  
    this.age = age;  
    this.hi = function () {  
        console.log(this.name, this.age);  
    };  
}
```

```
let ba = new Person("Ba", 28);  
let beo = new Person("Béo", 82);  
let bon = new Person("Bon", 3);  
  
ba.hi(); // "Ba"  
bon.hi(); // "Bon"
```



Homework

1. Viết hàm tạo Weapon với các thuộc tính name, damage, ...
2. Viết hàm tạo Player với các thuộc tính name, level, weapon, weapon object khởi tạo từ Weapon. Player có các methods:
 - attack() tính và in ra màn hình lượng sát thương gây ra = `weapon.damage * level`
 - changeWeapon() thay đổi weapon của Player
3. Tạo một vài đối tượng từ Player và Weapon, gọi phương thức attack() trên các đối tượng đó
4. Tạo một object calculator, có các thuộc tính a, b là 2 số, và các phương thức:
 - a. get(a, b) tham số nhận vào là 2 số a, b
 - b. add() trả về tổng $a + b$
 - c. sub() trả về hiệu $a - b$
 - d. div(), mul(), rem(), exp(), ... trả về kết quả phép tính tương ứng
 - e. Làm thế nào để cho phép gọi hàm theo chuỗi VD `cal.get(4,5).add()`; // 9

Homework

1. Viết hàm tạo Counter, có thuộc tính count, và các phương thức
 - count là tham số truyền vào, mặc định = 0
 - up() tăng count lên 1
 - down() giảm count 1
 - get() in ra count hiện tại
 - Làm thế nào để cho phép gọi hàm theo chuỗi VD `counter.up().up().get().down().get();`
2. Viết hàm tạo Girl có các thuộc tính và phương thức tùy ý 😄
3. Viết hàm tạo Boy có các thuộc tính và phương thức tùy ý 😊 (Boys, you know what I mean?)

