

Higher order function

Tham khảo thêm: [Chinh phục High Order Function, Closures, Currying và Callback trong Javascript](#)

High order function là một **function** mà nhận vào tham số là **1** hay nhiều **function** khác như đối số hoặc return về một **function** khác.

```
function formalGreeting() {  
  console.log("How are you?");  
}  
  
function casualGreeting() {  
  console.log("What's up?");  
}  
  
function greet(greetFormal, greetCasual) {  
  greetFormal();  
  greetCasual();  
}  
  
greet(formalGreeting, casualGreeting);
```

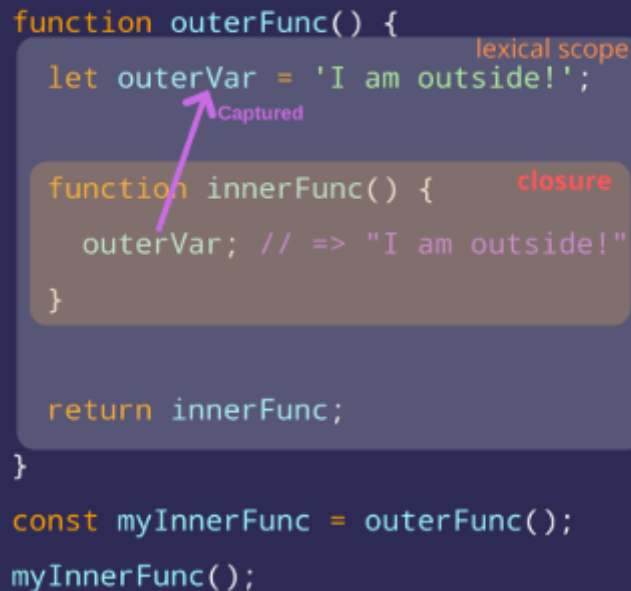
Callback function

Callback function là một **function** mà được truyền vào một **function khác** như một tham số

```
// Hàm nhận một số và một callback function để thực hiện phép tính bất kỳ  
  
function calculate(num, callback) {  
  // Thực hiện phép tính và gọi callback function với kết quả  
  const result = num * 2;  
  callback(result);  
}  
  
// Hàm callback để in ra kết quả  
function printResult(result) {  
  console.log('Kết quả là:', result);  
}  
  
// Gọi hàm calculate và truyền vào một số và hàm printResult như một callback  
calculate(5, printResult);
```

Closure

The **closure** captures variables from lexical scope



```
function outerFunc() {
  let outerVar = 'I am outside!';
  function innerFunc() {
    outerVar; // => "I am outside!"
  }
  return innerFunc;
}
const myInnerFunc = outerFunc();
myInnerFunc();
```

The diagram illustrates how a closure captures variables from its lexical scope. It shows a function `outerFunc` that defines a local variable `outerVar` and an inner function `innerFunc`. The inner function `innerFunc` references `outerVar`, which is captured by the closure. A pink arrow points from the text "Captured" to the `outerVar` variable. The text "lexical scope" is placed above the `outerVar` declaration, and "closure" is placed above the `innerFunc` function definition.

Source: <https://dmitripavlutin.com/javascript-closure/>

Closure là cách mà một **function cha** return về một **function con** bên trong nó. Ở trong **function con** đó có thể truy cập và thực thi các biến của **function cha** hoặc **function ông nội** mặc dù hàm cha đã dừng và return rồi. Phải đủ 2 điều kiện này mới được gọi là **Closure**.

```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init

  function displayName() { // the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();
```

```
function createCounter(initValue = 0) {
  let value = initValue; // private variable
  function increase() {
    value++;
  }
  function decrease() {
```

```
    value--;
  }
  function getValue() {
    return value;
  }
  return {
    getValue,
    increase,
    decrease,
  };

const counter = createCounter();
counter.getValue(); // 0
counter.increase();
counter.increase();
counter.getValue(); // 2
counter.decrease();
counter.getValue(); // 1
console.log(counter.value); // undefined
}
```

Currying

Currying là một **kỹ thuật** mà cho phép chuyển đổi một **function nhiều tham số** thành những **function liên tiếp có một tham số**.

Ví dụ:

```
sum(1)(2); // 3
function sum(x) {
  return function (y) {
    return x + y;
  };
}
```

Kết hợp với kỹ thuật Closure

```
// generate increase id
function createIdGenerator(startId = 1) {
  let id = startId;
  return function() {
    return id++;
  }
}

const getNextId = createIdGenerator(10);
getNextId(); // 10
getNextId(); // 11
getNextId(); // 12
```

Ứng dụng thực tế giữa **callback function** và **currying**

```
function findNumberLess10AndOdd() {
  const result = [];
  for(let i = 0; i < 10; i++) {
    if(i % 2 !== 0) {
      result.push(i);
    }
  }
  return result;
}
console.log(findNumberLess10AndOdd()); // --> Output: [1, 3, 5, 7, 9]

function findNumberLess20AndEven() {
  const result = [];
  for(let i = 0; i < 20; i++) {
    if(i % 2 === 0) {
      result.push(i);
    }
  }
  return result;
}
console.log(findNumberLess20AndEven()); // --> Output: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

function findNumberLess30AndDivide3surplus2() {
  const result = [];
  for(let i = 0; i < 30; i++) {
    if(i % 3 === 2) {
      result.push(i);
    }
  }
  return result;
}
console.log(findNumberLess30AndDivide3surplus2()); // --> Output: [2, 5, 8, 11, 14, 17, 20, 23, 26, 29]
```

Dùng callback function

```
function findNumber(num, func) {
  const result = [];
  for(let i = 0; i < num; i++) {
    if(func(i)) {
      result.push(i);
    }
  }
  return result;
}
console.log(findNumber(10, (num) => num % 2 !== 0)); // --> Output: [1, 3, 5, 7,
```

```
9]
console.log(findNumber(20, (num) => num % 2 === 0)); // --> Output: [0, 2, 4, 6,
8, 10, 12, 14, 16, 18]
console.log(findNumber(30, (num) => num % 3 === 2)); // --> Output: [2, 5, 8, 11,
14, 17, 20, 23, 26, 29]
```

Dùng currying

```
const findNumber = (num) => (func) => {
  const result = []
  for (let i = 0; i < num; i++) {
    if (func(i)) {
      result.push(i)
    }
  }
  return result
}
findNumber(10)((number) => number % 2 === 1)
findNumber(20)((number) => number % 2 === 0)
findNumber(30)((number) => number % 3 === 2)
```