



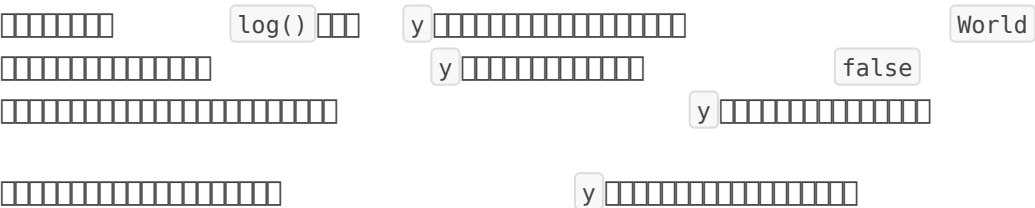
-  _____
 -  _____
 -  _____
 -  _____
 -  length  _____
 -  _____
 -  _____
- rest  _____
-  _____
- name  _____
-  _____
 -  _____
 -  _____
 -  _____
 -  _____
-  _____
 -  _____
 -  _____
 -  _____
 -  _____
 -  _____
 -  _____
-  _____
- Function.prototype.toString()
- catch  _____



ES6

```
function log(x, y) {
  y = y || 'World';
  console.log(x, y);
}

log('Hello') // Hello World
log('Hello', 'China') // Hello China
log('Hello', '') // Hello World
```



```
if (typeof y === 'undefined') {
  y = 'World';
}
```

ES6

```
function log(x, y = 'World') {
  console.log(x, y);
}

log('Hello') // Hello World
log('Hello', 'China') // Hello China
log('Hello', '') // Hello
```

ES6 ES5

```
function Point(x = 0, y = 0) {
  this.x = x;
}
```

```

    this.y = y;
  }

  const p = new Point();
  p // { x: 0, y: 0 }

```

ES6

```

// ...
// ...
// ...

```

```

// ...
let const

```

```

function foo(x = 5) {
  let x = 1; // error
  const x = 2; // error
}

```

```

// ...
x // ...
let const

```

```

// ...

```

```

// ...
function foo(x, x, y) {
  // ...
}

// ...
function foo(x, x, y = 1) {
  // ...
}

// SyntaxError: Duplicate parameter name not allowed in this context

```

```

// ...
// ...

```

```

let x = 99;
function foo(p = x + 1) {
  console.log(p);
}

foo() // 100

```

```
x = 100;
foo() // 101
```



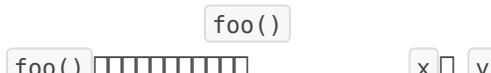





```
function foo({x, y = 5}) {
  console.log(x, y);
}

foo({}) // undefined 5
foo({x: 1}) // 1 5
foo({x: 1, y: 2}) // 1 2
foo() // TypeError: Cannot read property 'x' of undefined
```



```
function foo({x, y = 5} = {}) {
  console.log(x, y);
}

foo() // undefined 5
```






```
function fetch(url, { body = '', method = 'GET', headers = {} }) {
  console.log(method);
}

fetch('http://example.com', {})
// "GET"
```

```
fetch('http://example.com')  
  
// []
```

Diagram illustrating a sequence of memory blocks. The first block is labeled "fetch()". This is followed by a long sequence of blocks, with the last block labeled "fetch()".

```
function fetch(url, { body = '', method = 'GET', headers = {} } = {}) {  
  console.log(method);  
}  
  
fetch('http://example.com')  
// "GET"
```

The diagram illustrates the structure of a RESTful API request. It consists of the following components in sequence:

- A 6-character path (represented by a box divided into 6 segments).
- A `fetch` method (represented by a rounded rectangle).
- A 24-character body (represented by a long box divided into 24 segments).
- A `method` label (represented by a rounded rectangle).
- A `GET` label (represented by a rounded rectangle).
- A 24-character body (represented by a long box divided into 24 segments).

```
function f({ a, b = 'world' } = { a: 'hello' }) {  
  console.log(b);  
}  
  
f() // world
```

Diagram illustrating memory layout:

- Row 1: A 10-unit block, a function pointer `f()` pointing to a 20-unit block, and a JavaScript object `{ a: 'hello' }` with a pointer `b` pointing to a 5-unit block.
- Row 2: A 20-unit block.
- Row 3: A 20-unit block.

```
// 测试
function m1({x = 0, y = 0} = {}) {
  return [x, y];
}

// 测试
function m2({x, y} = { x: 0, y: 0 }) {
  return [x, y];
}

// 测试测试测试测试
```

```

m1() // [0, 0]
m2() // [0, 0]

// x □ y □□□□□□
m1({x: 3, y: 8}) // [3, 8]
m2({x: 3, y: 8}) // [3, 8]

// x □□□y □□□□□□
m1({x: 3}) // [3, 0]
m2({x: 3}) // [3, undefined]

// x □ y □□□□□□
m1({}) // [0, 0];
m2({}) // [undefined, undefined]

m1({z: 3}) // [0, 0]
m2({z: 3}) // [undefined, undefined]

```



```

// □□
function f(x = 1, y) {
  return [x, y];
}

f() // [1, undefined]
f(2) // [2, undefined]
f(, 1) // □□
f(undefined, 1) // [1, 1]

// □□
function f(x, y = 5, z) {
  return [x, y, z];
}

f() // [undefined, 5, undefined]

```

```
f(1) // [1, 5, undefined]
f(1, ,2) // []
f(1, undefined, 2) // [1, 5, 2]
```



undefined



```
function foo(x = 5, y = 6) {
  console.log(x, y);
}

foo(undefined, null)
// 5 null
```



x



y



length



length

```
(function (a) {}).length // 1
(function (a = 5) {}).length // 0
(function (a, b, c = 5) {}).length // 2
```



length



3



c

length

3

1

2



length



rest

length

```
(function(...args) {}).length // 0
```



length

```
(function (a = 0, b, c) {}).length // 0
(function (a, b = 1, c) {}).length // 1
```



context

```
var x = 1;

function f(x, y = x) {
  console.log(y);
}

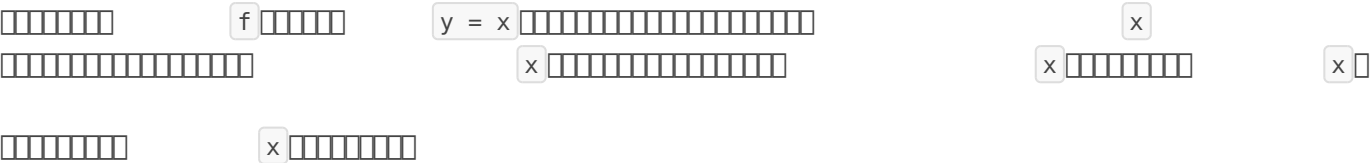
f(2) // 2
```



```
let x = 1;

function f(y = x) {
  let x = 2;
  console.log(y);
}

f() // 1
```



```
function f(y = x) {
  let x = 2;
  console.log(y);
}

f() // ReferenceError: x is not defined
```

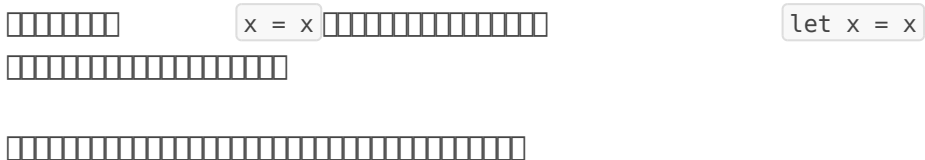


```
var x = 1;

function foo(x = x) {
    // ...
}

foo() // ReferenceError: Cannot access 'x' before initialization
```

```
foo() // ReferenceError: Cannot access 'x' before initialization
```

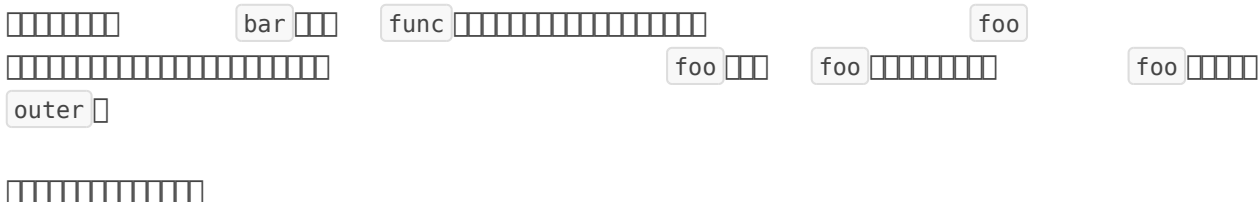


```
let foo = 'outer';

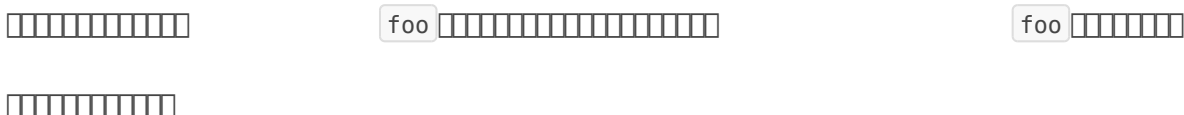
function bar(func = () => foo) {
  let foo = 'inner';
  console.log(func());
}

bar(); // outer
```

```
bar(); // outer
```



```
bar() // ReferenceError: foo is not defined
```



```
var x = 1;

function foo(x, y = function() { x = 2; }) {
  var x = 3;
```

```

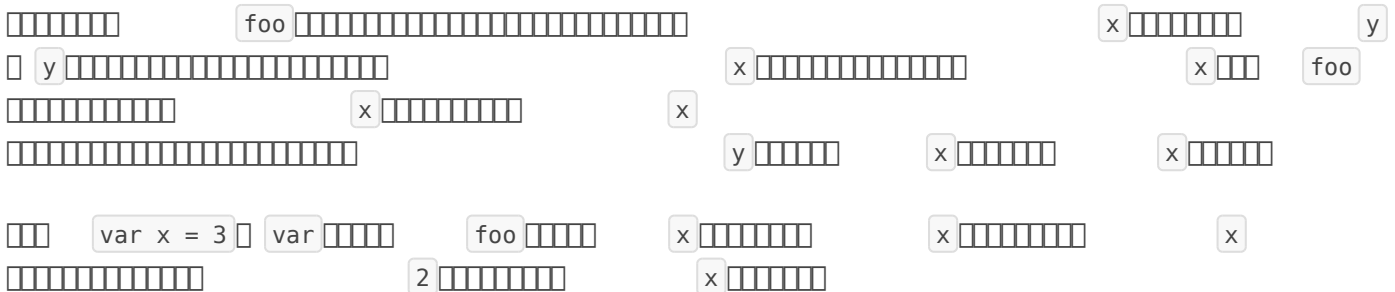
y();
console.log(x);
}

```

```

foo() // 3
x // 1

```



```

var x = 1;
function foo(x, y = function() { x = 2; }) {
  x = 3;
  y();
  console.log(x);
}

```

```

foo() // 2
x // 1

```



```

function throwIfMissing() {
  throw new Error('Missing parameter');
}

function foo(mustBeProvided = throwIfMissing()) {
  return mustBeProvided;
}

foo()
// Error: Missing parameter

```



throwIfMissing

[illegible]

undefined

```
function foo(optional = undefined) { ... }
```

rest

arguments

[illegible]

```
function add(...values) {
  let sum = 0;

  for (var val of values) {
    sum += val;
  }

  return sum;
}

add(2, 5, 3) // 10
```

```
add(2, 5, 3) // 10
```

rest []

arguments

--	--	--	--	--	--

```
// arguments[]
function sortNumbers() {
  return Array.from(arguments).sort();
}

// rest[]
const sortNumbers = (...numbers) => numbers.sort();
```

[illegible]

arguments 
 rest

 push 

Array.from

rest

```
function push(array, ...items) {  
  items.forEach(function(item) {  
    array.push(item);  
    console.log(item);  
  });  
}
```

```
var a = [];  
push(a, 1, 2, 3)
```

 rest 

```
//   
function f(a, ...b, c) {  
  // ...  
}
```

 length  rest 

```
(function(a) {}).length // 1  
(function(...a) {}).length // 0  
(function(a, ...b) {}).length // 1
```



 ES5 

```
function doSomething(a, b) {  
  'use strict';  
  // code  
}
```

ES2016

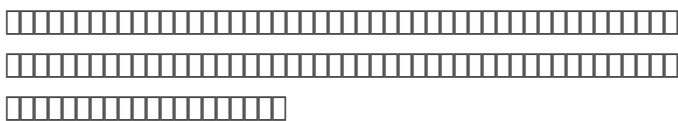



```
// 函数
function doSomething(a, b = a) {
  'use strict';
  // code
}

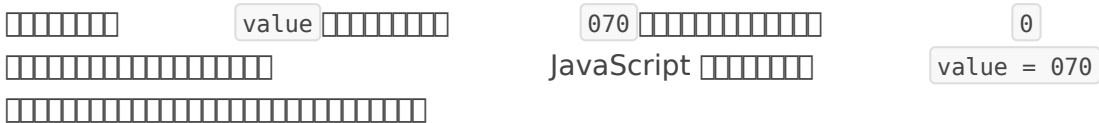
// 函数
const doSomething = function ({a, b}) {
  'use strict';
  // code
};

// 函数
const doSomething = (...a) => {
  'use strict';
  // code
};

const obj = {
  // 函数
  doSomething({a, b}) {
    'use strict';
    // code
  }
};
```



```
// 函数
function doSomething(value = 070) {
  'use strict';
  return value;
}
```



██
██

██

```
'use strict';

function doSomething(a, b = a) {
  // code
}
```

██

```
const doSomething = (function () {
  'use strict';
  return function(value = 42) {
    return value;
  };
})();
```

name

name

```
function foo() {}
foo.name // "foo"
```

ES6

ES6 ES5

```
var f = function () {};

// ES5
f.name // ""

// ES6
f.name // "f"
```

function

f function

ES5 ES6 name function

function

ES5 ES6 name function

```
const bar = function baz() {};
```

```
// ES5
```

```
bar.name // "baz"
```

```
// ES6
```

```
bar.name // "baz"
```

Function function

name function

anonymous

```
(new Function).name // "anonymous"
```

bind function

name function

bound function

```
function foo() {};
```

```
foo.bind({}).name // "bound foo"
```

```
(function({})).bind({}).name // "bound "
```

function

function

ES6 ES5 "function" => function

```
var f = v => v;
```

```
// ES5
```

```
var f = function (v) {
```

```
  return v;
```

```
};
```

function

```
var f = () => 5;
// 匿名函数
var f = function () { return 5 };

var sum = (num1, num2) => num1 + num2;
// 匿名函数
var sum = function(num1, num2) {
  return num1 + num2;
};
```

内存地址: 00000000000000000000000000000000

return 0000

```
var sum = (num1, num2) => { return num1 + num2; }
```

内存地址: 00000000000000000000000000000000

```
// 匿名函数
let getItem = id => { id: id, name: "Temp" };

// 匿名函数
let getItem = id => ({ id: id, name: "Temp" });
```

内存地址: 00000000000000000000000000000000

```
let foo = () => { a: 1 };
foo() // undefined
```

内存地址: 00000000000000000000000000000000

{ a: 1 } 内存地址: 00000000000000000000000000000000

a: 1 内存地址: 00000000000000000000000000000000

a 内存地址: 00000000000000000000000000000000

1; 内存地址: 00000000000000000000000000000000

内存地址: 00000000000000000000000000000000

```
let fn = () => void doesNotReturn();
```

内存地址: 00000000000000000000000000000000

```
const full = ({ first, last }) => first + ' ' + last;

// 匿名函数
function full(person) {
  return person.first + ' ' + person.last;
}
```

```
const isEven = n => n % 2 === 0;
const square = n => n * n;
```

```
// 関数定義
[1,2,3].map(function (x) {
    return x * x;
});

// 関数呼び出し
[1,2,3].map(x => x * x);
```

```
// 升序排列
var result = values.sort(function (a, b) {
    return a - b;
});

// 降序排列
var result = values.sort((a, b) => a - b);
```

```
const numbers = (...nums) => nums;

numbers(1, 2, 3, 4, 5)
// [1,2,3,4,5]

const headAndTail = (head, ...tail) => [head, tail];

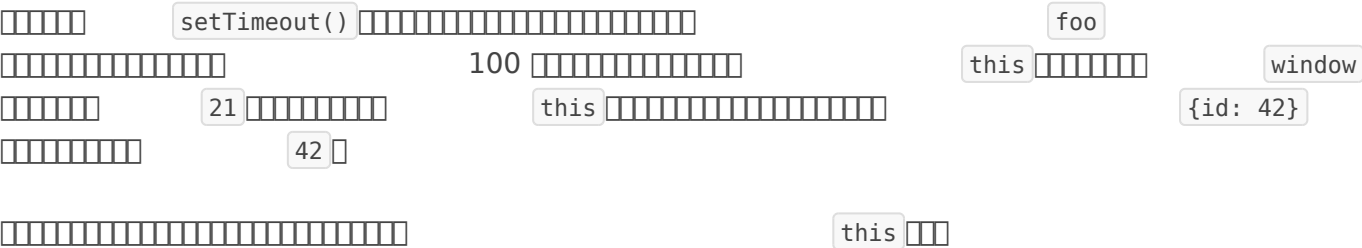
headAndTail(1, 2, 3, 4, 5)
// [1,[2,3,4,5]]
```



```
function foo() {
  setTimeout(() => {
    console.log('id:', this.id);
  }, 100);
}

var id = 21;

foo.call({ id: 42 });
// id: 42
```



```
function Timer() {
  this.s1 = 0;
  this.s2 = 0;
  // ...
  setInterval(() => this.s1++, 1000);
  // ...
  setInterval(function () {
    this.s2++;
  }, 1000);
}
```

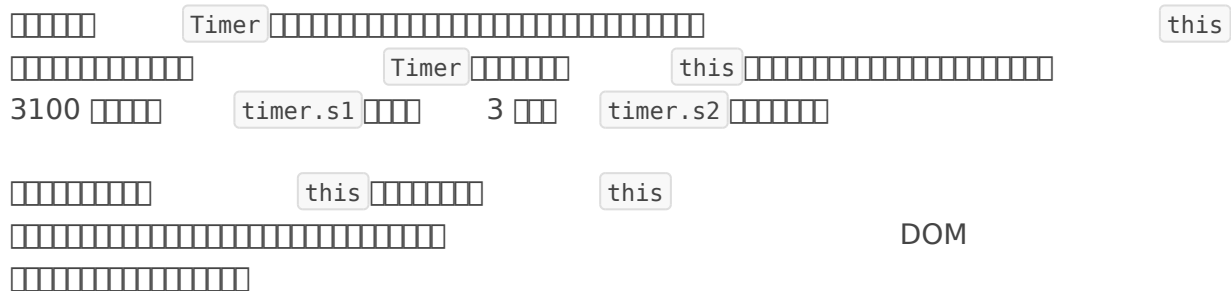
```

    }, 1000);
  }

  var timer = new Timer();

  setTimeout(() => console.log('s1: ', timer.s1), 3100);
  setTimeout(() => console.log('s2: ', timer.s2), 3100);
  // s1: 3
  // s2: 0

```



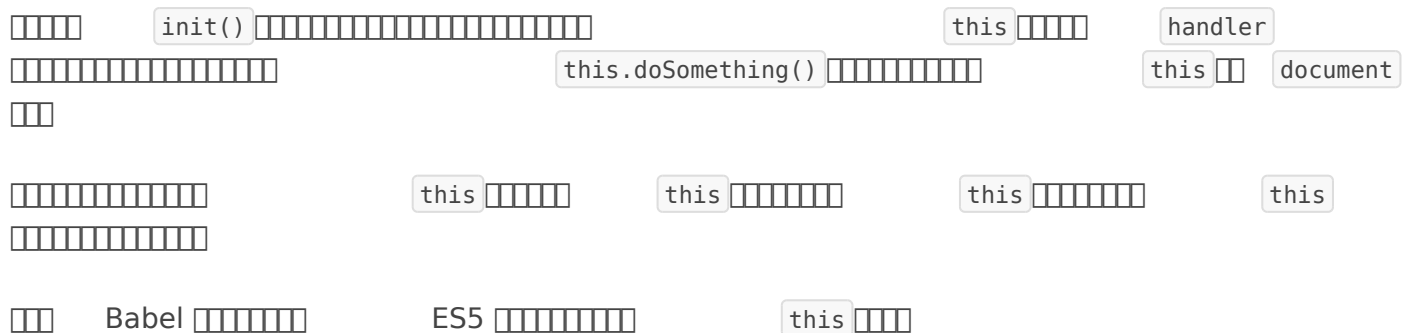
```

var handler = {
  id: '123456',

  init: function() {
    document.addEventListener('click',
      event => this.doSomething(event.type), false);
  },

  doSomething: function(type) {
    console.log('Handling ' + type + ' for ' + this.id);
  }
};

```



```

// ES6
function foo() {

```

```

setTimeout(() => {
  console.log('id:', this.id);
}, 100);

```

```

// ES5
function foo() {
  var _this = this;

  setTimeout(function () {
    console.log('id:', _this.id);
  }, 100);
}

```



```

function foo() {
  return () => {
    return () => {
      return () => {
        console.log('id:', this.id);
      };
    };
  };
}

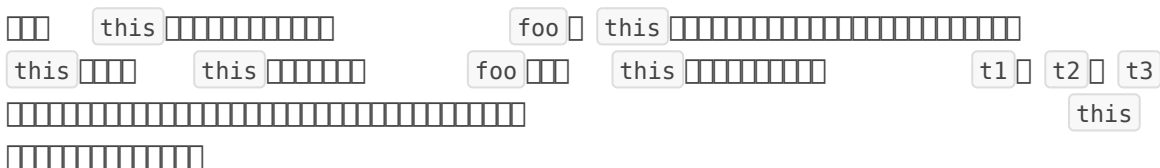
```

```
var f = foo.call({id: 1});
```

```

var t1 = f.call({id: 2})(); // id: 1
var t2 = f().call({id: 3})(); // id: 1
var t3 = f()().call({id: 4}); // id: 1

```



```
function foo() {
  setTimeout(() => {
    console.log('args:', arguments);
  }, 100);
}

foo(2, 4, 6, 8)
// args: [2, 4, 6, 8]
```

arguments

foo arguments

arguments

this

this

call() apply() bind()

this

```
(function() {
  return [
    (() => this.x).bind({ x: 'inner' })()
  ];
}).call({ x: 'outer' });
// ['outer']
```

this

bind

this

this

this

JavaScript

this

this

this

"this"

this

this

this " " " "

this

this

```
const cat = {
  lives: 9,
  jumps: () => {
    this.lives--;
  }
}
```

cat.jumps()

cat.jumps()

cat.jumps()

this

this

cat

this

jumps

jumps

--	--	--	--	--	--	--

```
globalThis.s = 21;

const obj = {
  s: 42,
  m: () => console.log(this.s)
};

obj.m() // 21
```

The diagram illustrates the memory layout of a JavaScript environment, divided into three main sections: Global, Heap, and Stack.

- Global:** Contains a single memory cell labeled `obj.m()`.
- Heap:** Contains multiple memory cells. One cell is labeled `obj.m` and contains the value `21`. Another cell is labeled `this` and contains the value `42`.
- Stack:** Contains multiple memory cells, some of which are labeled `obj.m()`.

Arrows indicate the flow of memory allocation and deallocation between these spaces.

```
globalThis.s = 21;

globalThis.m = () => console.log(this.s);

const obj = {
  s: 42,
  m: globalThis.m
};

obj.m() // 21
```

[illegible]

`this`

```
var button = document.getElementById('press');
button.addEventListener('click', () => {
  this.classList.toggle('on');
});
```

[illegible]



ES5 

```
function insert(value) {  
  return {into: function (array) {  
    return {after: function (afterValue) {  
      array.splice(array.indexOf(afterValue) + 1, 0, value);  
      return array;  
    }};  
  }};  
}  
  
insert(2).into([1, 3]).after(1); //[1, 2, 3]
```



```
let insert = (value) => ({into: (array) => ({after: (afterValue) => {  
  array.splice(array.indexOf(afterValue) + 1, 0, value);  
  return array;  
}})});  
  
insert(2).into([1, 3]).after(1); //[1, 2, 3]
```



pipeline

```
const pipeline = (...funcs) =>  
  val => funcs.reduce((a, b) => b(a), val);  
  
const plus1 = a => a + 1;  
const mult2 = a => a * 2;  
const addThenMult = pipeline(plus1, mult2);  
  
addThenMult(5)  
// 12
```



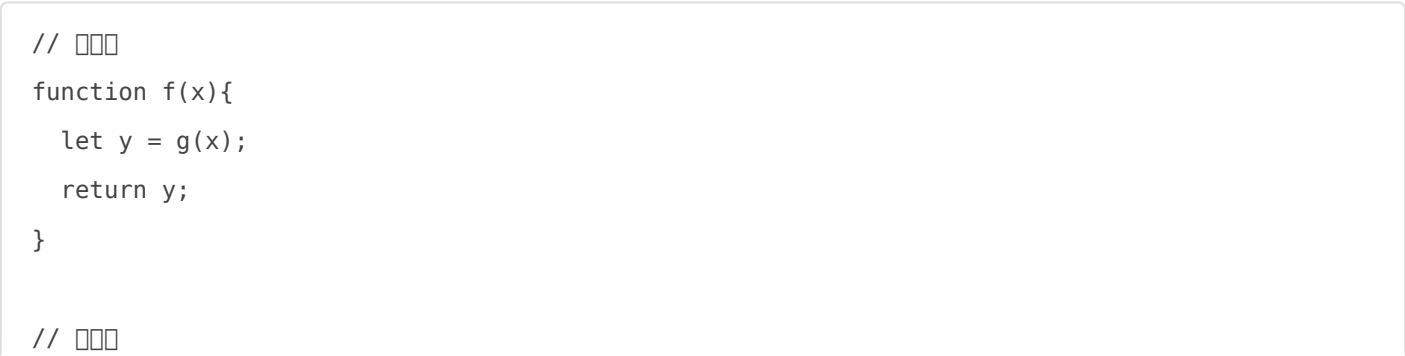
```
const plus1 = a => a + 1;  
const mult2 = a => a * 2;
```

```
mult2(plus1(5))  
// 12
```



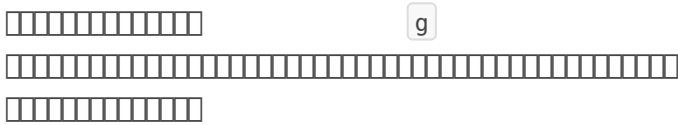
```
// λ-calculus
fix = λf.(λx.f(λv.x(x)(v)))(λx.f(λv.x(x)(v)))

// ES6
var fix = f => (x => f(v => x(x)(v)))
           (x => f(v => x(x)(v)));
```



```
function f(x){
  return g(x) + 1;
}
```

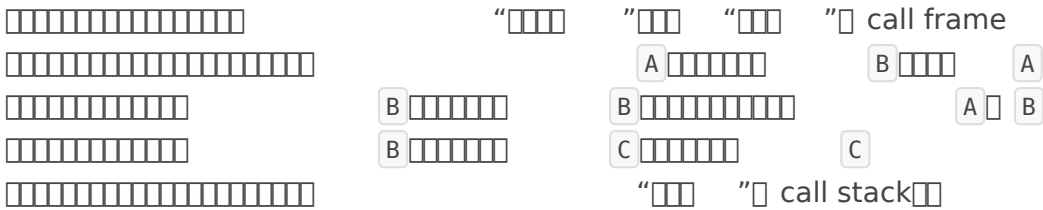
```
// 编译
function f(x){
  g(x);
}
```



```
function f(x){
  g(x);
  return undefined;
}
```



```
function f(x) {
  if (x > 0) {
    return m(x)
  }
  return n(x);
}
```



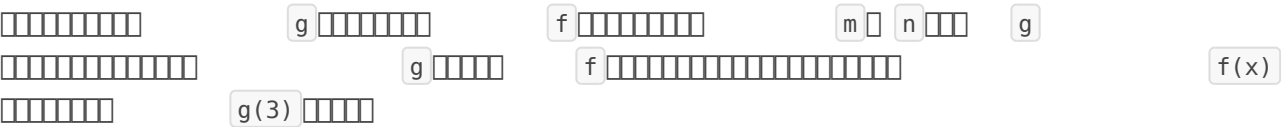
```
function f() {
  let m = 1;
  let n = 2;
  return g(m + n);
}

f();

// 编译
function f() {
  return g(3);
}

f();

// 编译
g(3);
```



“ ” Tail call optimization

“ ”

“ ”

```
function addOne(a){
  var one = 1;
  function inner(b){
    return b + one;
  }
  return inner(a);
}
```



Safari Chrome Firefox




```

return Fibonacci2 (n - 1, ac2, ac1 + ac2);
}

```

```

Fibonacci2(100) // 573147844013817200000
Fibonacci2(1000) // 7.0330367711422765e+208
Fibonacci2(10000) // Infinity

```

“” ES6
 ECMAScript “” ES6
 “”

“”

factorial total
 5 1 5

“”

```

function tailFactorial(n, total) {
  if (n === 1) return total;
  return tailFactorial(n - 1, n * total);
}

function factorial(n) {
  return tailFactorial(n, 1);
}

factorial(5) // 120

```

factorial tailFactorial
 currying
 “”

```

function currying(fn, n) {
  return function (m) {
    return fn.call(this, m, n);
  };
}

```

```
function tailFactorial(n, total) {
  if (n === 1) return total;
  return tailFactorial(n - 1, n * total);
}

const factorial = currying(tailFactorial, 1);

factorial(5) // 120
```

尾递归

tailFactorial

factorial

ES6

ES6

```
function factorial(n, total = 1) {
  if (n === 1) return total;
  return factorial(n - 1, n * total);
}

factorial(5) // 120
```

ES6

total

1

ES6

“

”

Lua ES6

ES6

ES6

ES6

- func.arguments
- func.caller

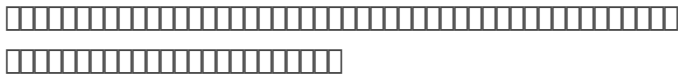
ES6

```
function restricted() {
  'use strict';
  restricted.caller; // 空
```

```

    restricted.arguments; // []
  }
  restricted();

```



```

function sum(x, y) {
  if (y > 0) {
    return sum(x + 1, y - 1);
  } else {
    return x;
  }
}

sum(1, 100000)
// Uncaught RangeError: Maximum call stack size exceeded(...)

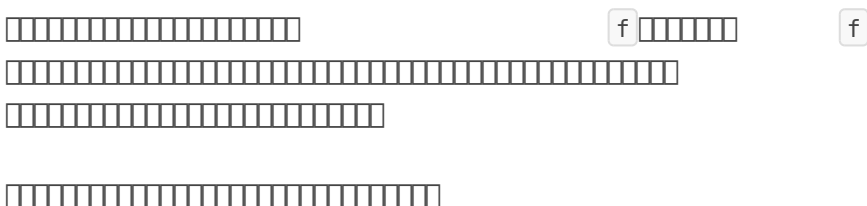
```



```

function trampoline(f) {
  while (f && f instanceof Function) {
    f = f();
  }
  return f;
}

```



```
function sum(x, y) {
  if (y > 0) {
    return sum.bind(null, x + 1, y - 1);
  } else {
    return x;
  }
}
```

|||||

sum |||||

|||||

sum |||||

```
trampoline(sum(1, 100000))
// 100001
```

|||||

```
function tco(f) {
  var value;
  var active = false;
  var accumulated = [];

  return function accumulator() {
    accumulated.push(arguments);
    if (!active) {
      active = true;
      while (accumulated.length) {
        value = f.apply(this, accumulated.shift());
      }
      active = false;
      return value;
    }
  };
}

var sum = tco(function(x, y) {
  if (y > 0) {
    return sum(x + 1, y - 1)
  }
  else {
    return x
  }
})
```

```

    }
  });

  sum(1, 100000)
  // 100001

```

```

    tco
    active
    sum undefined accumulated sum
    accumulator while
    " " " "

```

```

    [ ]

```

ES2017 [_](#) trailing comma

```

    [ ]

```

```

function clownsEverywhere(
  param1,
  param2
) { /* ... */ }

clownsEverywhere(
  'foo',
  'bar'
);

```

```

    param2 bar
    clownsEverywhere

```

```

function clownsEverywhere(
  param1,
  param2,
) { /* ... */ }

```

```
clownsEverywhere(  
  'foo',  
  'bar',  
);
```

Function.prototype.toString()

ES2019 ☐ ☐ ☐ ☐ ☐ ☐

toString()

toString()

```
function /* foo comment */ foo () {}

foo.toString()

// function foo() {}
```

```
// function foo() {}
```

```
// function foo() {}
```



`foo`

foo

```
toString()
```

```
function /* foo comment */ foo () {}

foo.toString()

// "function /* foo comment */ foo () {}"
```

```
// "function /* foo comment */ foo () {}"
```

```
// "function /* foo comment */ foo () {}"
```

catch

--	--	--	--	--	--

JavaScript ☐ ☐ ☐

try...catch

catch

try

[illegible]

```
try {
  // ...
} catch (err) {
  // 处理
}
```

try {
 // ...
} catch (err) {
 // ...
}

try {
 // ...
} catch {
 // ...
}

[ES2019](#)

```
try {  
  // ...  
} catch {  
  // ...  
}
```

try {	#2
// ...	
} catch {	2020-04-16 10:29:52 UTC
// ...	
}	2022-04-09 20:01:36 UTC