تورماهىرى js قوللانمىسى

javascript بولسا ئىنتىرنېتتا ئەڭ مودا بولغان بىر قوليازما تىلى بولۇپ،بۇ ئاساسلىق تورکۆرگۈچلەردە خىزمەت قىلىدۇ ،جۈملىدىن Firefox.Safair chrome ,Ie قاتارلىقلار,ئەلۋەتتە بۇيەردە بەك مۇرەككەپلەشۈرۈشتەمەسلىك ئۈچۈن تورکۆرگۈچتە قوللىنىلىدۇ دەپ يازدىم ئەمما،بۇنىڭ قوللىنىلىش دائىرىسى تور كۆرگۈچ بىلەنلا چەكلىنىپ قالمايدۇ،بۇ تىل يەنە مۇلازىمېتىر تەرەپ پىروگىراممىسى ئۈچۈنمۇ قوللىنىلىدىغان بولۇپ ھەر تۈردىكى سانلىق مەلۇمات ئامبارلىرى بىلەن ئالاقە قىلالايدۇ،جۈملىدىن mysql,mongodb ،بۇ تىل دۇنيادىكى ئەڭ چوڭ ئىجتىمائى ئوچۇق كود سۇپىسى بولغان github دىمۇ ھەرئايلىق داڭلىق تىللارنى باھالاشتا ئالدىنقى 10 تىلنىڭ ئىچىدىكى ئورنىنى ئۈزلۈكسىز ساقلاپلا قالماي يەنە داۋاملىق ئەڭ ئالدىغا قاراپ مېڭىۋاتىدۇ،بۇ تىل بىلەن بىز ھەرخىل مۇرەككەپ ھىسابلاش مەشغۇلاتلىرىنى بىر تەرەپ قىلالايمىز،جۈملىدىن نېرۋا سىستېمىسى ۋە ماششىنا تىلىنىمۇ ئۆز ئىچىگە ئالىدۇ، سىز بۇ تىل بىلەن پروگىراممىا يېزىشنى باشلاشدىن بۇرۇن چوقۇم HTML / XHTML نى بىلىشىڭىز كېرەك ،js بەتتىكى ھەربىر ئىلىمىنتى باشقۇرۇش خۇسۇسىيىتىغا ئىگە،دىمەكچىمەنكى javascript ئارقىلىق بەتتىكى ھەربىر dom ئىلىمىنتىنى خالىغانچە كونترول قىلالايسىز،dom دىمەكلىك document object model نىڭ قىسقارتىلمىسى بولۇپ سىز تورکۆرگۈدە ئاچقان پۈتۈن دۇنيادىكى ھەرقانداق بىر بىكەت دەل مۇشۇ dom بىلەن تۈزۈلگەن ئېلىمىنتلاردىن ئىبارەت،شۇنداقلا بۇ تىل يانفۇن ۋە كومپيۇتېرنىڭ قاتتىق دىتاللىرىغا كىرىش ئىقتىدارىغا ئىگە ،جۈملىدىن كامىرا،مىكروفون،قاتتىق دىسكىدىكى ھۆججەت قاتارلىقلار. مەقسەتكە كەلسەك بۇ قوللانمىنى يېزىشتىكى مەقسەت قانداقتۇر javascript نى يڭىدىن باشلىغۇچىلار ئۈچۈن پايدىلىنىش ماتېريالى قىلىش ئەمەس،بەلكى javascript دىكى دائىملىق خاتا چۈشەنچىلەر ۋە بىرقىسىم مۇرەككەپ مەسىلىلەرنى يېشىپ بېرىش،شۇڭلاشقا ئەگەر سىز javascript دىن ئاز تولا خەۋرىڭىز بولماي تۇرۇپ ،بۇ قوللانمىدىن پايدىلانماقچى بولسىڭىز كۆزلىگەن مەقسىتىڭىزگە ئېرىشەلمەسلىكىڭىز مومكىن،چۈنكى بۇ قوللانما javascript دىن ئوتتۇرھال سەۋىيەگە ئىگە بولغانلارغا ماس كېلىدۇ،بىشىدىن باشلىماقچى بولسىڭىز ئىنتىرنېتتىكى باشقا قوللانمىلارنى بىرقاتار كۆرۈڭ ۋە ئەمەلى سىناپ يېزىپ ئاساسى چۈشەنچىگە ئىگە بولۇڭ،ئاندىن بۇ قوللانما سىزگە ،سىز Javascript دا ئۈچىرىغان بېشىڭىزنى قايدۇرغان قېيىن مەسىلىلەرنى ھەل قىلىشتا زور ياردەمى تېگىدۇ.

ئوبيكت(object)

javascript دا هەممە نەرسە ئوبيكت،پەقەت Null ۋە undefined نى هسابقا ئالمىغاندا

'false.toString(); // 'false
'toString(); // '1,2,3.[3 ,2 ,1]

{}()function Foo
;Foo.bar = 1
Foo.bar; // 1

javascript دىكى دائىملىق بىر مەسىلە يەنى خاتالىق سان تىپى ئوبيكت شەكىلدە قوللىنىلمايدۇ،چۈنكى بۇ جاۋا سكرىپتىنىڭ ئاجىزلىقى بولۇپ ،سانىنى لەيلىمە سان float دەپ ئانالىز قىلىدۇ

2.toString(); // raises SyntaxError

بۇيەردە قىسمەن سانىنى ئويبكت شەكىلدە ئىشلىتىش ئۈسۈللىرىمۇ بار

2..toString(); // the second point is correctly recognized
2 .toString(); // note the space left to the dot
(2).toString(); // 2 is evaluated first

object سانلىق مەلۇمات تىپىغا ئوخشاش jsدىكى ئوبيكتمۇ hashmapsئوخشاش قوللىنىلىدۇ،بۇيەردىكى hashmap دىگىنىم java توپلىمىدا قوللىنىدىغان hashmap نى كۆرسىتىدۇ.ئوبيكت ئاساسلىقى خاسلىق ۋە مىتودتىن تەشكىل تاپقان بولۇپ قىممەتكە مەركەزلىشىدۇ بىز ئوبيكتنى {}بەلگىسى ئارقىلىق قۇرالايمىز،بۇيىڭى ئوبيكت Objecct.prototype قا ۋارىسلىق قىلىدۇ ۋە ئۆزىنىڭ ئىنقلانغان خاسلىقى بولمايدۇ.

var foo = {}; // a new empty object

// a new object with a 'test' property with value 12

```
var bar = {test: 12};
```

ئوبيكتنىڭ خاسلىقىغا كىرىش
ئوبيكتنىڭ خاسلىقىغا كىرىدىغان ئىككى خىل ئۇسۇل بار،بىرى چېكىت بىلەن،يەنى بىرى چاسا تىرناق بىلەن

```
var foo = {name: 'kitten'}
foo.name; // kitten
foo['name']; // kitten

var get = 'name';
foo[get]; // kitten

foo.1234; // SyntaxError
foo['1234']; // works
```

خاسلىققا كىرىشتىكى چېكىت بىلەن چاسا تىرناقنىڭ بىردىن بىر پەرقى شۇكى چاسا تىرناق بىلەن خاسلىققا كىرگەندە خاسلىقنىڭ ئىسمىنى يېزىشقا ۋە دىنامىكىلىق ھالدا خاسلىقنىڭ قىممىتىنى تەڭشەشكە بولىدۇ!

ئويبكتنىڭ خاسلىقنى ئۆچۈرۈش
جاۋا سكىرىپتتا ئوبيكتنىڭ خاسلىقنى delete مەشغۇلات بەلگىسى بىلەن ئۆچۈرۈشكە بولىدۇ،بۇ مەشغۇلات بەلگىسى خاسلىقتىكى قىممەتنى Null ياكى Undefined گە ئۆزگەرتىدۇ،ۋە قىممەتنى ئۆچۈردىدۇ،لىكىن ئاچقۇچلۇق سۆزنى ئەمەس

```
var obj = {
   bar: 1,
   foo: 2,
   baz: 3
};
obj.bar = undefined;
obj.foo = null;
delete obj.baz;

for(var i in obj) {
   if (obj.hasOwnProperty(i)) {
      console.log(i, " + obj[i]);
   }
}
```

ئۈستىدىكى كود bar نى undefined كە foo نى Null قىلىپ ئۆزىتىدۇ،پەقەت بۇيەردە baz نىڭ قىممىتىلا يوق،چۈنكى baz ئۆچۈرۈلگەن

تورماهىرى js قوللانمىسى

ئوبيكت ئاددى خەت شەكلى String ئوخشاش ھەرىپ بەلگىلەر بىلەن خاسلىق قورالايدۇ

```
var test = {
  'case': 'I am a keyword, so I must be notated as a string',
  delete: 'I am a keyword, so me too' // raises SyntaxError
};
```

يۇقارقى ئويبكتتىكى ئىككى خاسلىققا test.case ئارقىلىق كىرەلەيسىز،بەلكىم بەزى تور كۆرگۈلەر delete نى خاسلىق قىپ بىرىشنى قوللىماسلىقى مومكىن

Prototype (ئەندىزە)

javasccript نىڭ باشقا تىللارغا ئوخشاش object-oriented يەنى ئوبيكىتقا يۈزلەنگەن ئىقتىدارى يوق،بەلكىم بۇ Javascript نىڭ بىردىن بىر ئاجىزلىقى بولۇشى مومكىن لىكىن Javascript نىڭ Prototypal ۋارسلىق قىلىش ئىقتىدارى بار،بىز بۇنىڭ بىلەن ئوبيكىتقا يۈزلەنگەن پىروگىراممىا ئىقتىدارىنى ئەمەلگە ئاشۇرالايمىز،نۆۋەتتە javascript نىڭ ecmascript6 ئۆلچىمى ئەمىللەشمەكتە كەلگۈسى بىر يىلغا يەتمىگەن ئەھۋال ئاستىدا جاۋا سكرىپتتىا چوڭ ئىقتىدارلار ھازىرلىنىشى بەرھەق،مەقسەتكە كەلسەك javascript نىڭ ۋارسلىق قىلىش ئىقتىدارى كەڭ قوللىنىدىغان ئىقتىدار

```
function Foo() {
  this.value = 42;
}
Foo.prototype = {
  method: function() {}
};

function Bar() {}

// Set Bar's prototype to a new instance of Foo
Bar.prototype = new Foo();
Bar.prototype.foo = 'Hello World';

// Make sure to list Bar as the actual constructor
Bar.prototype.constructor = Bar;

var test = new Bar(); // create a new bar instance

// The resulting prototype chain
test [instance of Bar]
Bar.prototype [instance of Foo]
    { foo: 'Hello World' }
```

```
Foo.prototype
{ method: ... }
Object.prototype
 { toString: ... /* etc. */ }
```

يۇقارقى كودتا test ئوبيكتى Bar.prototype ۋە Foo.prototype غا ۋارسلىق قىلىپ foo دا ئىنقلانغان فۇنكسىيە مىتودىغا كىرەلەيدۇ

خاسلىق ئىزدەش
ئوبيكتنىڭ خاسلىقىغا كىرگەن ۋاقىتتا javascript تەلەپ قىلىنغان نام ئۈچىدكى خاسلىقنى تاپقانغاقەدەر ئەسلى تىپ زەنجىرىنى نى كىسىپ ئۆتىدۇ،ئەگەر ئەسلى تىپ زەنجىرىObject.prototypeغا يېتىپ ،بەلگىلەنگەن خاسلىقنى تاپالمىسا undefined قىممىتىنى قايتۇرىدۇ

```
function Foo() {}
Foo.prototype = 1; // no effect
```

ئەسلى تىپ خاسلىقى
ئەسلى تىپ خاسلىقى تىل تەرىپىدىن prototype chain ئەسلى تىپ زەنجىرى قۇرۇشقا ئىشلىتىلگەن ۋاقىتتا بۇنىڭغا ھەرقانداق قىممەتنى تەقسىملەش مومكىن،لىكىن javascript نىڭ ئېتىدائى تىپى string,number,boolean,float لارنى ئەسلى تىپ قىلىپ تەقسىملىگەندە رەت قىلىنىشى مومكىن

ئىجرا بولۇشى
ئەسلى تىپ زەنجىرىدە خاسلىق ئىزدىگەن ۋاقىتتا ،ئىزدەش ۋاقتى javascript كودلىرىنىڭ ئىجرا بولۇش سۈرئىتىگە بىسىم ئەپ كىلىدۇ،ئىھتىمال بۇ كودنىڭ ئىجرا بولۇش سۈرئىگە تەسىر كۆرسىتىدىغان ھالقىلىق مەسىلە بولۇشى مومكىن دىمەكچى Proptype chain (ئەسلى تىپ)،بۇنىڭغا قوشۇپ ئېيتقاندا ئەسلى مەۋجۇت بولمىغان خاسلىققا كىرمەكچى بولغىنىمىزدا javascript بارلىق ئەسلى تىپ زەنجىرىدىن خاسلىقنى تېپىش ئۈچۈن كىسىپ ئۆتىدۇ،ھەمدە ئوبيكتنىڭ ھەر بىر خاسلىقى ئۈستىدىن دەۋرىيلەش ئىلىپ بارغاندا ،ئەسلى تىپ زەنجىرى prototype chain مۇ تەكشۈرۈلىدۇ.
خۇلاسە:
ھەرقانداق مۇرەككەپ كود يىزىشتىن بۇرۇن javascript نىڭ Prototypal زەنجىرىنى چۈشىنىش ناھايىتى مۇھىم،ھەمدە كودلىرىڭىزدا prototype chain ئەسلى تىپ زەنجىرىنىڭ ئۇزۇنلىقىغا دىققە قىلىشىڭىز كىرەك،چۈنكى بۇ سىزنىڭ يازغان كودلىرىڭىزنىڭ ئىجرا بولۇش سۈرئىتىگە تەسىر كۆرسىتىدىغان بىردىن-بىر ئامىل بولۇشى مومكىن

hasownProperty(ئۆزىنىڭ خاسلىقى)تەرجىمە قىلىۋەتتىم بۇنىمۇ ھى ھى
قايسى ئوبيكتنىڭ ئۆزىدە قايسى خاسلىق ئىنقلانغان بۇنى تەكشۈرۈ ئۈچۈن hasownProperty نى ئىشلىتىمىز،
hasOwnPropertyمتودىنى ئىشلىتىش ناھايىتى موھىم يەنى ھەممە ئوبيكت Object.prototype قا ۋارسلىق قىلىدۇ Prototype جاۋاسكرىپتتىكى خاسلىق بىلەن مەشغۇلات قىلىشتا قوللىنىلىدىغان ،ئوبيكتنىڭ hasownProperty نى كىسىپ ئۆتىمەيدىغان بىردىن بىر نەرسە chain

```
Object.prototype.bar = 1;
var foo = {goo: undefined};
```

```
foo.bar; // 1
'bar' in foo; // true

foo.hasOwnProperty('bar'); // false
foo.hasOwnProperty('goo'); // true
```

پەقەت hasownProperty مىتودى توغرا ۋە كۆزلىگەندەك نەتىجە بىرىدۇ،ئوبيېكتنىڭ خاسلىقىنى تەكشۈرۈشتە بۇ مىتود بەكلا مۇھىم،بۇيەردە يەنى javascript دا ئوبيېكتنىڭ ئۆزىدە ئىنقلانمىغان خاسلىقتىن قۇتۇلۇشنىڭ ھېچقانداق چارىسى يوق،ئىھتىمال ئۇنىڭ Prototype chain سىدا بولۇشى مومكىن

for in Loop(ئوبيېكتنىڭ خاسلىقىنى دەۋرىيلەش)
پەقەت in مەشغۇلات بەلگىسىگە ئوخشاش،for in loop ئوبيېكتنىڭ خاسلىقى ئۈستىدىن دەۋرىيلەش ئىلىپ بارغاندا ئەسلى تىپ زەنجىرىنى ئاختۇرىدۇ

```
Object.prototype.bar = 1;

var foo = {moo: 2};
for(var i in foo) {
   console.log(i); // prints both bar and moo
}
```

Loop بەدىنى يەنى ئىچىدە ئوبيېكتنىڭ زۆرۈر بولمىغان خاسلىقىنى سۈزۈپ ئىلىۋىتىش كىرەك بولسا Object.prototype نىڭ hasownProperty مىتودى قوللىنىلىدۇ
hasOwnProperty نى ئىشلىتىپ سۈزۈش

```
for(var i in foo) {
```

فۇنكسىيە

دا تۈنجى تۈر ئوبيېكتى ھىسابلىنىدۇ،بۇ دىگەنلىك فۇنكىيسەنى باشقا فۇنكسيەلەرگە پارامىتر javascript فۇنكسيە شەكلىدە ئۆتكۈزۈش دەپ ئاتىلىدۇ callback شەكلىدە ئۆتكۈزگىلى بولىدۇ،ئادەتتە بۇخىل ئىشلىتىلىش نامسز فۇنكىيسەنى فۇنكسيەنى ئىنقلاش ئۈسۈلى

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

ئۈستىدكى كود پىروگىراممىا باشلىنىپ ئىجرا قىلىنىشتىن بۇرۇن ئۆزى ئىنقلانغان ھەرقانداق دائىر ئىچىدە كۈچكە ئىگە ،ئەگەر بىزبۇنى ھەقىقى ئىنقلاشتىن بۇرۇن چاقىرساقمۇ يەنىلا ئوخشاش خىزمەت قىلىۋىرىدۇ

```
foo(); // Works because foo was created before this code runs
function foo() {}
```

تورماهىرى js قوللانمىسى

ئېپادىلەش فۇنكسىيەسى
بۇخىل فۇنكسىيەدە بىز نامسىزفۇنكسىيەنى مقدار fooغا تەقسىملەپ بېرىمىز

```
foo; // 'undefined'
foo(); // this raises a TypeError
var foo = function() {};
```

var مقدار ئىنقلاش ئاچقۇچى بولغاچقا ھەقىقى كود ئىجرا بولۇشتىن foo ئىجرا بولىدۇ چۈنكى تەقسىملەش پەقەت ئىجرا مۇھىتىدىلا پەيدا بولىدۇ،fooنىڭ قىممىتى ماس كود ئىجرابولۇشتىن بۇرۇن ئەسلىدىنلا undefined يەنى ئىنقلانمىغان بولىدۇ

ئىسىملىق فۇنكسىيە ئېپادىلەش
فۇنكسىيەنىڭ باشقىچە بىرخىل ئېپادىلەش ئۇسۇلى

```
var foo = function bar() {
  bar(); // Works
}
bar(); // ReferenceError
```

this نىڭ ئالاھىدىلىكى
javascript دىكى this ھالقىلىق سۆزىنىڭ باشقا پىروگىراممىا تىلىرىدىكى this قا قارىغاندا باشقىچە ئۇقۇمى بار
Javscript دىكى this نىڭ باشقا پىروگىراممىا تىللىرىغا قارىغاندا 5 خىل ئوخشىماسلىق بار

ئومۇمى دائىرلىك يەنى (global) دائىرە
this ئومۇمى دائىرە(global)دائىرىسىدە ئىشلىتىلگەندە ئومۇمى دائىرلىك ئوبيېكتقا ئىشارەت قىلىدۇ

فۇنكسىيەنى چاقىرىش

```
foo();
```

مىتودنى چاقىرىش(يەنى ھەرىكەت)

```
test.foo();
```

بولىدۇ()test نىڭ ئىشارەت قىلىدىغىنى this بۇ مىسالدا

قۇرۇلمىلىك فۇنكسىيە

```
new foo();
```

فۇنكسىيە new ھالقىلىق سۆزى بىلەن چاقىرىلسا بۇ قۇرۇلمىلىق فۇنكسىيە بولىدۇ.بۇڭاقىتتا قۇرۇلمىلىق فۇنكسىيەنىڭ ئىچىدىكى this يېڭىدىن قۇرۇلغان ئوبيېكتقا ئىشارەت قىلىدۇ

```
function foo(a, b, c) {}
var bar = {};
foo.apply(bar, [1, 2, 3]); // array will expand to the below
foo.call(bar, 1, 2, 3); // results in a = 1, b = 2, c = 3
```

Function.prototype نىڭ مىتودى apply ۋە callئىشلىتىلگەن ۋاقىتتا,چاقىرىلغان فۇنكسىيەنىڭ ئىچىدىكى this نىڭ قىممىتى ئوچۇق ئاشكارا ھالدا،ماس كىلىدىغان فۇنكسىيە چاقىرىقىنىڭ بىرىنجى پارامىترغا تەڭشىلىدۇ

ئورتاق خاتالىق
گەرچە قارىماققا مەنىسى باردەك قىلسىمۇ ئەمما ،باشقا بىرخاتا لايھەلەنگەن تىل دەپ قاراشقا بولىدۇ،(دىمەكچىمەنكى javascript نىڭ ئازراق كەمچىلكلىرى بار)

دائىملىق بىر خاتالىق test نىڭ ئىچىدىكى this نىڭ foo غا ئىشارەت قىلىشىدۇر،ئەمىلىيەتتە ئۇنداق ئەمەس!
test نىڭ ئىچىدىن foo غا كىرىش ئۈچۈن،foo غا ئىشارەت قىلىدىغان مىتودنىڭ ئىچىدە يەرلىك ئۆزگەرگۈچى مىقدار قۇرۇش كىرەك

```
Foo.method = function() {
  var that = this;
  function test() {
    // Use that instead of this here
  }
  test();
}
```

that نورمال ئۆزگەرگۈچى مىقدار نامى،بىراق ھەمىشە بۇ سىرتقى this قا قوللانما قىلىشقا ئىشلىتىلىدۇ
مىتودنى تەقسىملەش
javascript دا ئىشلىمەيدىغان يەنە بىرنەرسە بولسا مىتودنى ئۆزگەرگۈچى مىقدارغا تەقسىملەش

```
var test = someObject.methodTest;
test();
```

بۇ ئەھۋالدا test()ئاددى فۇنكسىيە چاقىرىقىغا ئوخشاش ئىپادە بىلدۇرىدۇ،شۇڭلاشقا ئىچىدكى this ھالقىلىق سۆزى someobject قا ئىشارەت قىلالالمايدۇ

ئاخىرلاشتۇرۇش ۋە قوللانما(reference)
javascript نىڭ ئەڭ كۈچلۈك ئىقتىدارىنىڭ بىرى ئۇنىڭ ئاخىرلاشتۇرۇش(يەنى چەك دائىرىسىنى بىكىتىش)ئىقتىدارىنىڭ بولغانلىقىدۇر،دائىرە يەنى(scope)ئۆزى ئىنىقلانغان دائىرىدە ئىچىدە سىرتقى دائىرە(scope)غا كىرىشنى ساقلاپ قالىدۇ، javascript دا فۇنكسىيەنى دائىرىسى بولۇپ،ئەسلىدىنلا ھەممە فۇنكسىيە closure (يەنى ئاخىرلاشتۇرۇش،چىگىراسىنى بەلگىلەشكە)ئوخشاش خۇسۇسىيتى بار

خۇسۇسى مقدارنى تەقلىدلەش

```
function Counter(start) {
  var count = start;
  return {
    increment: function() {
      count++;
    },

    get: function() {
      return count;
    }
  }
}

var foo = Counter(4);
foo.increment();
foo.get(); // 5
```

بۇ كودتا Counter دائىرە closure (يەنى چەك دائىرىسى)بولغان ئىككى دانە increment ۋە get فۇنكسىيەسىنى قايتۇرىدۇ،بۇ فۇنكسىيەنىڭ ئىككىلىسى counter نىڭ دائىرىسىگە(scope)غا بىر قوللانمىنى ساقلاپ قالىدۇ،شۇڭلاشقا count ئۆزگەرگۈچى مقدارغائۆزى ئىنقلانغان دائىرە ئىچىدە كىرگىلى بولىدۇ

خۇسۇسى مقدارنىڭ خىزمەت پىرىنسىپى
javascript دائىرەنى تەقسىملەش ياكى ئۇنى قوللانما قىلىش مۇمكىن بولمىغانلىقى ئۈچۈن،count قا تىشىدىن كىرىشنىڭ ھېچقانداق چارىسى يوق،پەقەت ئۇنىڭ بىلەن ئالاقە قىلىش ئۇسۇلى بولسا ئىككى دانە closure ئارقىلىق بولىدۇ

```
var foo = new Counter(4);
foo.hack = function() {
  count = 1337;
};
```

ئۇستىدىكى كود Counter نىڭ دائىرەسىدىكى مقدار count نى ئۆزگەرتمەيدۇ،چۈنكى ئۇ دائىرەدە foo.hack ئىنقلانمىغان،بۇنىڭ ئورنىدا بىر بولسا global دائىرلىك مقدار ئۇستى-ئۇستىلەپ يىزىلىدۇ ياكى قۇرۇلىدۇ

loop نىڭ بەدىندىكى closure
دائىم سادىر قىلىنىدىغان خاتالىق بولسا loops نىڭ ئىچىدە closure (چەكلىمە)نى قوللىنىش

```
for(var i = 0; i < 10; i++) {
  setTimeout(function() {
    console.log(i);
  }, 1000);
```

```
}
```

ئۇستىدىكى كود 0 دىن تارتىپ 9 غىچە قىممەت چىقارمايدۇ،بىراق 10 نى 10 قېتىم چىقىرىدۇ،نامسىز فۇنكسىيە i غا قارىتا قوللانمىنى ساقلاپ قالىدۇ،بۇ ۋاقىتتا console.log چاقىرىلىدۇ،for loop دەۋرىيلەش تاماملىنىدۇ ۋە i نىڭ قىممىتى 10 غا تەڭشىلىدۇ

قوللانما قىلىش مەسىلىسىدىن ساقلىنىش

```
for(var i = 0; i < 10; i++) {
  (function(e) {
    setTimeout(function() {
      console.log(e);
    }, 1000);
  })(i);
}
```

نامسىز سىرتقى فۇنكسىيە i بىلەن بىرگە ئۇنى بىرىنجى پارامېتىر شەكلىدە دەرھاللا چاقىرلىدۇ،ئاندىن i نى ئۆزىنىڭ پارامېترى e شەكلىدە i نىڭ كۆچۈرۈلمىسىنى قوبۇل قىلىدۇ
نامسىز فۇنكسىيەنىڭ setTimeout قا ئۆتكۈزۈلگەن e غا قارىتا قوللانمىسى بولۇپ،بۇ قىممەتلەر loop تەرىپىدىن ئۆزگەرتىلمەيدۇ،بۇيەردە بۇخىل مەقسەتكە يېتىشنىڭ باشقىچە ئۇسۇللىرىمۇ بار،بۇنىڭ ئۈسۈلى بولسا بىر فۇنكسىيەنى نامسىز فۇنكسىيەدىن قايتۇرۇش بولۇپ،ئاندىن يۇقارقىدىكى كودتىكىگە ئوخشاش ئۇنۇمگە ئىرىشكىلى بولىدۇ

```
for(var i = 0; i < 10; i++) {
  setTimeout((function(e) {
    return function() {
      console.log(e);
    }
  })(i), 1000)
}
```

بۇيەردە يەنە bind ئىشلىتىپ مۇشۇ خىل مەقسەتكە يېتىشنىڭ ئۇسۇلى بار بولۇپ،بۇخىل ئۇسۇلدا this نىڭ كونتېكستى ۋە پارامېترى فۇنكسىيەگە باغلىنىدۇ،ئاندىن يۇقارقىدىكى كودقا ئوخشاش نەتىجە ھاسىل بولىدۇ

```
for(var i = 0; i < 10; i++) {
  setTimeout(console.log.bind(console, i), 1000);
}
```

arguments ئوبيېكتى
جاۋاسكرىپتتىكى ھەربىر فۇنكسىيە دائىرىسى arguments دەپ ئاتالغان ئالاھىدە مقدارغا كىرەلەيدۇ،بۇ مقدار فۇنكسىيەگە ئۆتكۈزۈلگەن بارلىق arguments نىڭ تىزىملىكىنى كونترول قىلىدۇ مۇھىتىم نوقتا شۇكى arguments نىڭ ئۆزى array (سانلار گۇرۇپپىسى ئەمەس)بىراق بۇمقدارنىڭ array(سانلار گۇرۇپپىسىنىڭكىگە)ئوخشاش length خاسلىقى بار،بۇ Array.prototype دىن ۋارسلىق قوبۇل قىلمىغان بولۇپ،ئەمەلىيەتتە بۇ بىر ئوبيېكت ھىسابلىنىدۇ،مۇشۇ

سەۋەبلىك ئۆلچەملىك سانلار گۇرۇپپىسى مىتودلىرى push, pop ياكى slice نى arguments دا ئىشلەتكىلى بولمايدۇ ئۆلچەملىك سانلار گۇرۇپپىسى مىتودىنى ئىشلىتىش ئۈچۈن arguments نى array غا ئايلاندۇرۇش كىرەك.ئاستىدكى بۇ كود arguments ئوبيكتىنىڭ بارلىق ئىلمىتلىرنى ئۆزئىچگە ئالغان يېڭى سانلار گۇرۇپپىسى(array)نى قايتۇرىدۇ

```
if (foo.hasOwnProperty(i)) {
 console.log(i);
   }
}
```

---

array(سانلار گۇرۇپپىسى)

گەرچە سانلار گۇرۇپپىسىمۇ ئوبيكت بولسىمۇ بىراق for in نى ئىشلىتىشنىڭ ياخشىراق سەۋەبى يوق،بىراق سانلار گۇرۇپپىسىغا قارشى ئىتلىشنىڭ نورغۇنلىغان ياخشى سەۋەپلىرى بار،چۈنكى for in تىپ زەنجىرىدىن بارلىق خاسلىقلارنى ئىزدەيدىغان بولۇپ ئەڭ ياخشى hasownproperty نى ئىشلەتكەن ياخشى چۈنكى hasownproperty نىڭ سۈرئىتى تولىمۇ تىز ___ دەۋرىلەش
سانلارگۇرۇپپىسنى دەۋرىلەشنىڭ ئەڭ ياخشى ئۇسۇلى كىلاسسىك for نى ئىشلىتىش

```
var list = [1, 2, 3, 4, 5, ..... 100000000]; for(var i = 0, l = list.length; i < l; i++)
{ console.log(list[i]); }
```

length خاسلىقى بىز array دىكى ئىلمىنتلارغا ئىرىشكىنىمىزدە length خاسلىقى سانلار گۇرۇپپىسىدىكى بارلىق ئىلمىنتلارنىڭ سانىنى قايتۇرىدۇ،لىكىن بىز length خاسلىقى بىلەن سانلار گۇرۇپپىسىدىكى ئىلمىنتلارنى بەلگىلىنىمىزدە سانلار گۇرۇپپىسىدىكى ئىلمىنتلارنى كىسۈتىدۇ

```
var foo = [1, 2, 3, 4, 5, 6];
foo.length = 3;
foo; // [1, 2, 3]

foo.length = 6;
foo.push(4);
foo; /
```

كېچىك بىر length نى بىكىتىش بىلەن array دىكى ئىلمىنت سانى تۆۋەنلەيدۇ،لىكىن سانلار گۇرۇپپىسىنىڭ بىكارلىنىشى ئىشپ ماڭىدۇ! خۇلاسە سۈرئىتىنىڭ تىز بولۇشى ئۈچۈن سانلار گۇرۇپپىسىنى دەۋرىلىگەندە ئەڭ ياخشىسى for نى ئىشلىتىش ھەمدە خاسلىقى length ساقلاش،ئەمما array(سانلار گۇرۇپپىسىدا)for in ئىشلەتكەنلىك ناچار كود يازغانلىقىنىڭ ۋە چوقۇم چاتاق چىقىدىغانلىقىنىڭ ئىپادىسى!

array قۇرۇلمىسى(constructor)
گەرچە array قۇرۇلمىسى پارامتر بىلەن ياردى كەلدى قىلىشتا ناھايىتى ياخشى بولسىمۇ بىراق تەۋسىيە قىلىنمىدىغان ئۇسۇل ،خەچە []بەلگىسى بىلەن array قۇرۇشتۇر

```
[1, 2, 3]; // Result: [1, 2, 3]
new Array(1, 2, 3); // Result: [1, 2, 3]

[3]; // Result: [3]
new Array(3); // Result: []
new Array('3') // Result: ['3']
```

ئۇستىدىكى كودتا array قۇرۇلمىسىغا پەقەت بىردانە پارامتىر ئۆتكۈزۈلگەن،ئۆتكۈزۈلگەن پارامتىر سان بولغان ۋاقىتتا،بۇ قۇرۇلما length خاسلىقى پارامتىر خاسلىقىغا بەلگىلەنگەن بىر شالاڭ(قۇرۇق array(نى قايتۇرىدۇ،بۇيەردە دىققەت قىلىدىغان نوقتا شۇكى يېڭى array نىڭ length خاسلىقى بۇ ئۇسۇلدا بىكتىلسە،array نى ئەمەلى ئىندەكسلەش (يەنى سانلار گۇرۇپپىسىدىكى ئىلمىتنلارنىڭ ئورۇن نومۇرى)قوزغىتىلمىغان بولىدۇ

```
var arr = new Array(3);
arr[1]; // undefined
1 in arr; // false, the index was not set
```

array غا length خاسلىقىنى يۇقىرى دەرىجىدە بىكىتىش ئۈچۈن پايدىلىق ئۇسۇل بولسا string تەكرارلاش بولۇپ،بۇ دەۋرىلەشتىن تۇىسىدۇ

```
new Array(count + 1).join(stringToRepeat);
```

خۇلاسە
array قۇرۇلمىسىنى قۇرۇش ئۈچۈن خەتچە [] بەلگىسىنى ئىشلىتىش ئەڭ ياخشى كۆرۈلگەن ئۇسۇل بولۇپ،بۇ ئۇسۇلدا كود گىراممىاتىكىسى پاكىزە چىقىدۇ،ھەمدە كودنىڭ چۈشىنىشچانلىقىنى يۇقىرى كۆتۇرىدۇ!

تىپلار

---

js تا ئوبيېكت قىممىتىن سىلىشتۇرۇشنىڭ ئىككى خىل ئۇسۇلى بار تەڭلەشتۇرۇش بەلگىسى تەڭلەشتۇرۇش بەلگىسى ئىككى دانە تەڭلىمەدىن تەركىب تاپقان:== javasscript نىڭ ئىقتىدارى ئاجىزلىقى تىپلاشتۇرۇش بولۇپ،بۇنداق دىگەنلىك قىممەتنى سىلىشتۇرۇش ئۈچۈن زورلۇق قوللىنىلىدۇ

```
""          ==   "0"          // false
0           ==   ""           // true
```

تورماهىرى js قوللانمىسى

```
0          ==   "0"          // true
false      ==   "false"      // false
false      ==   "0"          // true
false      ==   undefined    // false
false      ==   null         // false
null       ==   undefined    // true
" \t\r\n"   ==   0
```

ئۇستىدكى جەدۋەل تىپ مەجبۇرلاشنىڭ نەتىجىسنى كۆرسىتىپ بىرىدۇ،بۇنىڭ ئاساسلىق سەۋەبى بولسا == تەڭلەشتۈرۈش بەلگىسنى قوللىنىش كەڭ دائىرلىق ناچار ئۇسۇل دەپ قارالغان،ھەمدە بۇ خىل ئۇسۇلدا سلىشتۈرۈش ئىپ كۆپ مىقداردىكى كودتىن خاتالىقنى تېپىشنى قىينلاشتۇرۇپ قويىدۇ،ھەمدە بۇخىل ئۇسۇلدا سلىشتۈرۈش ئىپ بىرىشنىڭ كودنىڭ ئىجرا بولۇش سۈرئىتىگە تەسىرى بار

چىڭ ھالەتتىكى تەڭلەشتۈرۈش بەلگىسى
چىڭ ھالەتتىكى تەڭلەشتۈرۈش بەلگىسى ئۇچ دانە === بەلگىدىن تەركىپ تاپقان بۇخىلدىكى تەڭلەشتۈرۈش بەلگىسى نورمال ھالەتتە تەڭلەشتۈرۈش ئىپ بارالايدۇ،بۇندىن باشقا تېپقا مەجبۇرلاش قوللانمايدۇ

```
""         ===   "0"          // false
0          ===   ""           // false
0          ===   "0"          // false
false      ===   "false"      // false
false      ===   "0"          // false
false      ===   undefined    // false
false      ===   null         // false
null       ===   undefined    // false
" \t\r\n"   ===   0           // false
```

ئۇستىدكى جەدۋەلدىكى كودنىڭ نەتىجسى ناھايتى ئىنىق،بۇخىل ئۇسۇلدا تەڭلەشتۈرۈش ئىپ بارغاندا كودنىڭ پۈختلىقنى مۇئەييەن دەرىجىدە ئاشۇرىدۇ

ئوبيېكتنى سلىشتۈرۈش

== ۋە === ئىككىلسى ئوخشاشلا تەڭلەشتۈرۈش بەلگىسى دەپ ئاتىلىدۇ،ئەمما بۇ تەڭلەشتۈرۈش بەلگىرى ئوبيېكت(object)تە قوللىنىلغاندا ئوخشمايدۇ

```
{} === {};                  // false
new String('foo') === 'foo'; // false
new Number(10) === 10;      // false
var foo = {};
```

```
foo === foo;        // true
```

ئۇستىدكى ئىككىلى تەڭلەشتۈرۈش بەلگىسى پەرىقنى سلىشتۈرىدۇ،تەڭلىكنى ئەمەس،ئۇلار ئوبيېكتنىڭ ئوخشاش ئۇلگىسى(instance)سلىشتۈرىدىغان بولۇپ،بۇ python دىكى is ۋە c دىكى كۆرسەتكۈچ(pointer)غا ئوخشايدۇ
خۇلاسە
تەۋسىيە قىلىنغان تەڭلەشتۈرۈش ئۇسۇلى بولسا === نى ئىشلىتىش == نى ئەمەس

مەشغۇلات بەلگىسى typeof

---

typeof مەشغۇلات بەلگىسى ۋە instanceof مەشغۇلات بەلگىسى javascript لايىھەسىنىڭ ئەڭ چوڭ كەمچىلىكلىرى بولۇشى مومكىن ،ھەمدە بۇلار پۈتۈنلەر بۇزۇلاي دەپ قالدى،گەرچە instanceof مەشغۇلات بەلگىسىنىڭ ئىشلىتىش چەكلىمىسى بولسىمۇ،بىراق typeof نىڭ يەنىلا ئەملى كارغا ياراىدىغان جايلىرىمۇ بار
javascript تىپلار جەدۋىلى

Value	Class	Type
"foo"	String	string
new String("foo")	String	object
1.2	Number	number
new Number(1.2)	Number	object
true	Boolean	boolean
new Boolean(true)	Boolean	object
new Date()	Date	object
new Error()	Error	object
[1,2,3]	Array	object
new Array(1, 2, 3)	Array	object
new Function("")	Function	function
/abc/g	RegExp	object (function in Nitro/V8)
new RegExp("meow")	RegExp	object (function in Nitro/V8)
{}	Object	object
new Object()	Object	object

ئۇستىدكى جەدۋەلدىكى تىپلار typeof مەشغۇلاتى قايتۇرىدىغان قىممەتلەرنى كۆرسىتىدىغان بولۇپ،تىپلار ھەرقانداق خالىغان بىر تىپ بولسا بولمۇبرىدۇ،بىراق بۇ مۇقىم، Class ئوبيېكتنىڭ ئىچكى [class]خاسلىق قىممىتىنى كۆرسىتىدۇ، class نىڭ قىممىتىنى قايتۇرۇش ئۈچۈن Object.prototype نىڭ toString()ى متودىنى ئىشلىتىشكە بولىدۇ
ئوبيېكتنىڭ تۈرى
javascript تۈر[[class]]قىممىتىگە كىرىش ئۈچۈن،Object.prototype.toString متودىنى ئىشلىتىش ئۇسۇلىنى بەرگەن

تورماهىرى js قوللانمىسى

```
function is(type, obj) { var clas = Object.prototype.toString.call(obj).slice(8, -1); return obj !== undefined && obj !== null && clas === type; } is('String', 'test'); // true is('String', new String('test')); // true
```

ئۇستىدىكى مىسالدا،Object.prototype.toStrin مۇشۇ [[class]] قىممىتى قايتۇرۇلدىغان ئوبيېكتقا بەلگىلەنگەن (مەركەزلەشكەن)this نىڭ قىممىتى بىلەن چاقىرىلغان
ئىنقلانمىغان مىقدارنى تەكشۈرۈش

```
typeof foo !== 'undefined'
```

بۇ مىسالدا foo ھەقىقى ئىنقلانغانمۇ يوق تەكشۈرىدۇ،پەقەت ئۇنى قوللانما قىلىش ReferenceError خاتالىقىنى كەلتۈرۈپ چىقىرىدۇ،بۇ typeof نىڭ بىردىن بىر پايدىلىق تەرىپى بولۇشى مومكىن. خۇلاسە ئوبيېكت تىپىنى تەكشۈرۈشتە تەۋسىيە قىلىنىدىغان ئۇسۇل Object.prototype.toString مىتودىنى قوللىنىش،چۈنكى بۇ مۇشۇنداق قىلىشنىڭ ئەڭ ياخشى ئۇسۇلى،ئۇستىدكى تىپلار جەدۋىلدە كۆرسىتىلگەندەك، typeof قايتۇرىدىغان بەزى تىپلار javascript نىڭ ئىشلىتىش قائىدىسىدە ئىنقلانمىغان،شۇڭا بۇنى ئەمەلى قوللانغاندا پەرىق يۈزبېرىدۇ،مەيلى قانداقلا تىپىنى تەكشۈرۈش بولمىسۇن typeof نى قوللىنىشتىن ساقلىنىش كېرەك!
