Javascript

flow control

for

```
for(var i = 0; i < len; i++)
{
  state1;
}
```

```
for(var k in Obj){
  // if
Obj.hasOwnProperty(k)
{
    console.log('Obj[' + k
+ '] = ' + Obj[k]);
  // }
}
```

if

```
false ? a : b
```

```
if(cond){
  state1;
}else if(cond2){
  state2;
}else{
  state3;
}
```

try catch finally

```
var a = {};
try{
  a.f();
}catch(e){
  console.log(e
instanceof
TypeError);    // true
  (function(){

console.log('a.f()');      //
'a.f()'
  }}();
}finally{

console.log('always');
// 'always'
}
```

while

```
while(cond){
  state1;
}
```

```
do {
  state1;
} while(cond)
```

```
break; continue
```

switch

```
switch (day) {
  case MON:
    break;
  case TUE:
    break;
  case WEN:
    break;
  default:
}
```

Identifiers

- Case-sensitive
  - a != A
- start with letters
  - ascii, unicode
- key words
  - break; case; catch; continue; default; delete; do; else; finally; for; function; if; in; instanceof; new; return; switch; this; throw; try; typeof; var; void; while; with
- variables
  - global — a = 123
  - local — var a = 123
- reserved words
  - non-strict
    - class; const; enum; export; extends; import; super;
  - 'use strict'
    - implements; interface; let; package; private; protected; public; static; yield;

Data Types

Numbers

- type of — 'number'
- value
  - 0
  - 1.2
  - 045
  - 0x56
  - 0.314e2
  - NaN
- Number()
  - parseInt("2.7", 10); // 2
  - parseInt('10',2); // 2
  - parseFloat('2.5'); // 2.5
  - isNaN(parseFloat('a')): // true

Boolean

- typeof — 'boolean'
- value
  - true
  - false
- Boolean()
  - number
    - Boolean(0); // false
    - Boolean(NaN); // false
  - string
    - Boolean(!0); // true
    - Boolean(''); // false
    - Boolean('*'); // true
  - object
    - Boolean(null); // false
    - **Boolean({}); // true**
  - undefined
    - Boolean(undefined); // false

undefined

- typeof — 'undefined'
- value — undefined
- undefined != Not defined

```
undefined == undefined;  // true
var a;
a == undefined;     // true
console.log(d);     // Error
```

Null //

- typeof — 'object'
- value — null
- 'undefined object'
  - null == undefined; // true

Object

- typeof — 'object'
- { }
  - literal
    - var p = { name: 'Bob', 'age': 24};
  - constructor
    - var q = {};
      q.name = 'Bob';
      q.age = 24;

      console.log(q.name);   // 'Bob'
      console.log(q['age']);   // 24
- new Object()

```
var p   = new Object();
p.sayHi = function(){
  console.log('Hi!');
};
p.sayHi();      // 'Hi!'
```

Function

- typeof — 'function'

```
var a = function(arg1, arg2)
{
  //func body;
}
```

- create
  - function a(arg1, arg2)
    {
    //func body;
    }
  - (function(arg1,arg2)
    {
    //func body;
    })
- inside function
  - arguments
    - arguments
      - var f = function(){
        console.log(arguments);
        };
        f(3,2,1);   // { '0': 3, '1': 2, '2': 1 }
    - undefined
      - var f = function(x){
        console.log(x);
        };
        f();   // undefined
    - // by order
      - var f = function(x, y){
        console.log('x: ' + x + ', y: ' + y);
        };
        var x = 'x', y = 'y';
        f(x, y);   // x: x, y: y
        f(y, x);   // x: y, y: x

scope

- scope chain
- closure
- execution context

Operators

- + - * / %
  - + // string > number > boolean
  - castiing
    - - // number > string > boolean
- a++ vs. ++a
- logical
  - ! not
    && and
    || or
- comparison
  - < <= > >= != ==
    - == for value
      === for value and type

Reference Type

object

- constructor

```
function Person(name){
  this.name = name;
  this.sayHi= function(){
    console.log('Hi ' + this.name + '!');
  };
};

var p1 = new Person('Ad');
var p2 = new Person('Bob');

p1.name       // 'Ad'
p2.sayHi();      // 'Hi Bob!'

p1.constructor === Person  // true
p1 instanceof Person   // true
p1.sayHi == p2.sayHi   // false
```

- inheritance
  - constructor

```
function Father(){
  this.familyName = 'Good';
};

function Child(){
  Father.call(this);
};

var c = new Child();
console.log(c.familyName);     // 'Good'
```

  - prototype chain

```
function Father(){};
Father.prototype.familyName = 'Good';

function Child(){};
Child.prototype = new Father();

var c = new Child();
console.log(c.familyName);    // 'Good'
```

- prototype chain

```
function Person(){};
Person.prototype.name = 'Person';

var p1  = new Person();
var p2  = new Person();
p2.name = 'rainy';

console.log(p1.name);     // 'Person'
console.log(p2.name);     // 'rainy'
```

  - root of everything

```
// All properties on the prototype are shared among instances
Person.prototype.friends = ['Ad', 'Bob'];

p1.friends.pop();     // 'Bob'
console.log(p2.friends);    // ['Ad']

// Combine constructor & prototype
function Person(){
  this.friends = ['Ad', 'Bob'];
};
Person.prototype.name = 'Person';
```

```
p1 instanceof Person       // true
p1.age = 24;

p1.hasOwnProperty('name');    // false
Person.hasOwnProperty('name');  // true

p1.hasOwnProperty('age');    // true
'name' in p1        // true
'age' in Person       // false
```

array

var a = new Array(1, 2, '3', [4, true]);

- typeof — 'object'
- properties — a.length // 4
- iteration
  - for(var i = 0; i < a.length; i++){
    console.log(a[i]);
    }
  - a.forEach(function(ele){
    console.log(ele);
    });
- methods
  - a.slice(0.2); // [1,2]
  - a.indexOf(1); // 0
  - a.push({ });
    // return a.length(Mod)
  - a.pop();
    // return popped element(Mod)
  - a.toString() == a.join(',');
  - a.join('array');
    - var s = 'a,b,c,d';
      s.split(',');
      // ['a', 'b', 'c', 'd']
  - shift
    unshift
    sort
    reverse
    splice
  - filter
    some
    every
  - a.concat(['a', 'b']); // VS push()
  - map/reduce

```
// arr.map(callback, thisObj)

/* callback = function(element, index, arr){
 *   return element to the same pos of (returned)arr;
 * };
 */

[55, 44, 33, 22, 11].map(function(e, i, arr){
  return e/(arr.length-i);
});        // [ 11, 11, 11, 11, 11 ];
```

```
// arr.reduce(callback, init);

/* callback = function(prev, curr, index, arr){
 *    curr walk through
 *      arr.slice(init != undefined ? 0 : 1, arr.length);
 *    prev cache last returned value start with:
 *      (init != undefined ? init : arr[0]);
 * };
 */

[55, 44, 33, 22, 11].reduce(function(p, c, i, arr){
  return p + c/(arr. length-i);
}, 55);      // 0;
```

function

- as value (callback)
  - var api = function(){
    return {
      name: 'rainy',
      age: 24
    };
    };

    var handler = function(d){
      console.log('Name: '+d.name+', Age: '+d.age);
    };

    var request = function(api, callback){
      callback(api());
    };

    request(api, handler); //
    Name: rainy,
    Age: 24
- typeof — 'object'
- properties — .length
- methods
  - apply
    call
    bind
    - // func.apply(thisObj, [arg1, arg2, ...]);
      // func.call(thisObj, arg1, arg2, ...);
      // func.bind(thisObj, arg1, arg2, ...);
      // about thisObj, see `Scope`
    - var sayHi = function(name){
      console.log('Hello, ' + name + '!');
      };

      sayHi.call(this, 'rainy');   // Hello, rainy!
      sayHi.apply(this, ['rainy']);   // Hello, rainy!
      sayHi.bind(this, 'rainy')();   // Hello, rainy!