1. 创建10个 a 标签，点击的时候弹出来对应的序号

```
for(var i=0;i<10;i++){
    (function(i){
        var a=document.createElement('a');
        a.innerHTML=i+'<br>';
        document.body.appendChild(a);
        a.addEventListener('click',function(e){
            e.preventDefault();  //取消默认事件，指a标签
            alert(i);
        });
    })(i);
}
```

2. 请你实现一个深克隆

```
var deepCopy = function(obj) {
    if (typeof obj !== 'object') return;
    var newObj = obj instanceof Array ? [] : {};
    for (var key in obj) {
        if (obj.hasOwnProperty(key)) {
            newObj[key] = typeof obj[key] === 'object' ? deepCopy(obj[key]) : obj[key];
        }
    }
    return newObj;
}
```

3. 谈谈你对模块化开发的理解

所谓的模块化开发就是封装细节，提供使用接口，彼此之间互不影响，每个模块都是实现某一特定的功能。模块化开发的基础就是函数。

模块化的实现 nodejs 中的 module.exports与require

require.js 中的AMD规范

sea.js 中的CMD规范

es6 中的export与import

4. 请问如何解决在滚动事件中需要做个复杂计算或者实现一个按钮的防连续点击操作?

使用防抖或者节流来实现

```
    //防抖
    function debounce(func, wait, immediate) {
        var timeout, result;
        var debounced = function () {
            var context = this;
            var args = arguments;
```

```
            if (timeout) clearTimeout(timeout);
            if (immediate) {
                // 如果已经执行过，不再执行
                var callNow = !timeout;
                timeout = setTimeout(function(){
                    timeout = null;
                }, wait)
                if (callNow) result = func.apply(context, args)
            }
            else {
                timeout = setTimeout(function(){
                    func.apply(context, args)
                }, wait);
            }
            return result;
        };
        debounced.cancel = function() {
            clearTimeout(timeout);
            timeout = null;
        };
        return debounced;
    }
```

```
    //节流
    function throttle(func, wait, options) {
        var timeout, context, args, result;
        var previous = 0;
        if (!options) options = {};
        var later = function() {
            previous = options.leading === false ? 0 : new Date().getTime();
            timeout = null;
            func.apply(context, args);
            if (!timeout) context = args = null;
        };
        var throttled = function() {
            var now = new Date().getTime();
            if (!previous && options.leading === false) previous = now;
            var remaining = wait - (now - previous);
            context = this;
            args = arguments;
            if (remaining <= 0 || remaining > wait) {
                if (timeout) {
                    clearTimeout(timeout);
                    timeout = null;
                }
                previous = now;
                func.apply(context, args);
                if (!timeout) context = args = null;
            } else if (!timeout && options.trailing !== false) {
                timeout = setTimeout(later, remaining);
            }
        };
```

```
        return throttled;
    }
```

5. 为什么 0.1 + 0.2 != 0.3

因为 JS 采用 IEEE 754 双精度版本（64位），并且只要采用 IEEE 754 的语言都有该问题，我们都知道在计算机中都是以二进制来存储的，0.1和0.2的二进制是：

```
    // (0011) 表示循环
    0.1 = 2^-4 * 1.10011(0011)
    0.2 = 2^-3 * 1.10011(0011)
```

由于刚才我们说了JS采用IEEE 754双精度。用六十四位代表一个数，六十四位中符号位占一位，整数位占十一位，其余五十二位都为小数位。因为 0.1 和 0.2 都是无限循环的二进制了，所以在小数位末尾处需要判断是否进位（就和十进制的四舍五入一样）。

所以 2^-4 * 1.10011...001 进位后就变成了 2^-4 * 1.10011(0011 * 12次)010 。那么把这两个二进制加起来会得出 2^-2 * 1.0011(0011 * 11次)0100 , 这个值算成十进制是 0.30000000000000004，0.30000000000000004当然不等于0.3。

6. 请验证用户名，用户名规则，以字母开头中间可以数字下划线 6-20个字符

使用正则表达式

```
/^[a-zA-Z][a-zA-Z0-9_]{5,19}$/
```

7. 请模拟实现一个bind函数

```
Function.prototype.myBind = function (context) {
  if (typeof this !== 'function') {
    throw new TypeError('Error')
  }
  var _this = this
  var args = [...arguments].slice(1)
  // 返回一个函数
  return function F() {
    // 因为返回了一个函数，我们可以 new F()，所以需要判断
    if (this instanceof F) {
      return new _this(args, ...arguments)
    }
    return _this.apply(context, args.concat(arguments))
  }
}
```

8. ES6中的promise有了解吗？可以说一下它是做什么的吗？

Promise 是 ES6 新增的语法，解决了回调地狱的问题。

可以把 Promise 看成一个状态机。初始是 pending 状态，可以通过函数 resolve 和 reject ，将状态转变为 resolved 或者 rejected 状态，状态一旦改变就不能再次变化。then 函数会返回一个 Promise 实例，并且该返回值是一个新的实例而不是之前的实例。因为 Promise 规范规定除了 pending 状态，其他状态是不可以改变的，如果返回的是一个相同实例的话，多个 then 调用就失去意义了。

最好可以说一下源码实现。

9. 红灯三秒亮一次，绿灯两秒亮一次，黄灯一秒亮一次；如何让三个灯不断交替重复亮灯？三个亮灯函数已经存在：

```
function red(){
    console.log('red    - ', new Date());
}
function green(){
    console.log('green  - ', new Date());
}
function yellow(){
    console.log('yellow - ', new Date());
}
```

主要考察多种回调函数写法

1. callback实现

```
    function loop(){
        setTimeout(function(){
            red();
            setTimeout(function(){
                green();
                setTimeout(function(){
                    yellow();
                    loop();
                }, 1000)
            }, 2000)
        }, 3000);
    }
    loop();
```

2. Promise实现

```
    function tic(timer, callback){
        return new Promise(function(resolve, reject){
            setTimeout(function(){
                callback();
                resolve();
            }, timer)
        })
    }
    var promise = new Promise(function(resolve, reject){ resolve() });
    function loop(){
        promise.then(function(){
```

```
            return tic(3000, red);
        }).then(function(){
            return tic(2000, green);
        }).then(function(){
            return tic(1000, yellow);
        }).then(function(){
            loop();
        })
    }
    loop();
```

3. Generator实现

```
    function* light(){
        yield tic(3000, red);
        yield tic(2000, green);
        yield tic(1000, yellow);
    }

    function loop(iterator, gen){
        var result = iterator.next();
        if (result.done) {
            loop(gen(), gen)
        } else {
            result.value.then(function(){
                loop(iterator, gen)
            })
        }
    };

    loop(light(), light);
```

4. async, await实现

```
    (async function (){
        while(true){
            await tic(3000, red);
            await tic(2000, green);
            await tic(1000, yellow);
        }
    })();
```