Homework

Javascript
Function

Content

Params	Callback	Scope	Closure
Context	Change Context	Arrow Function	Render

Default Parameter

```
function multiply(a, b) {
    return a * b
}
multiply(5, 2) // 10
multiply(5) // NaN

function multiply(a, b) {
    b = (b === undefined) ? b : 1;
    return a * b
}
multiply(5) // 5
```

```
function multiply(a, b = 1) {
    return a * b;
}

multiply(5) // 5

multiply(5, undefined) // 5

multiply(5, null) // 0
```

Careful with object parameter

```
const myCar = {
    userName: "Honda",
    color: "black"
}
function decorate(car){
    car.color = "red";
}
decorate(myCar);
console.log(myCar);
// { userName: 'Honda', color:
'red' }
```

Arguments

```
function sum(a, b) {
    return a + b;
}

function sum(a, b, c) {
    console.log(arguments)
    return a + b + c;
}

sum(1, 2, 3);
// [Arguments]{ '0': 1, '1': 2, '2': 3 }
// have length, no array method
```

```
function sum() {
    const len = arguments.length;
    let total = 0;
    for (let i = 0; i < len; i++) {
        total += arguments[i];
    }
    console.log(total);
    return total;
}

sum(1, 2, 3); // 6
sum(1, 2, 3, 4, 5); // 15
```

Rest parameter

```
function sum() {
    const len = arguments.length;
    let total = 0;
    for (let i = 0; i < len; i++) {
        total += arguments[i];
    }
    console.log(total);
    return total;
}
sum(1, 2, 3); // 6
sum(1, 2, 3, 4, 5); // 15
```

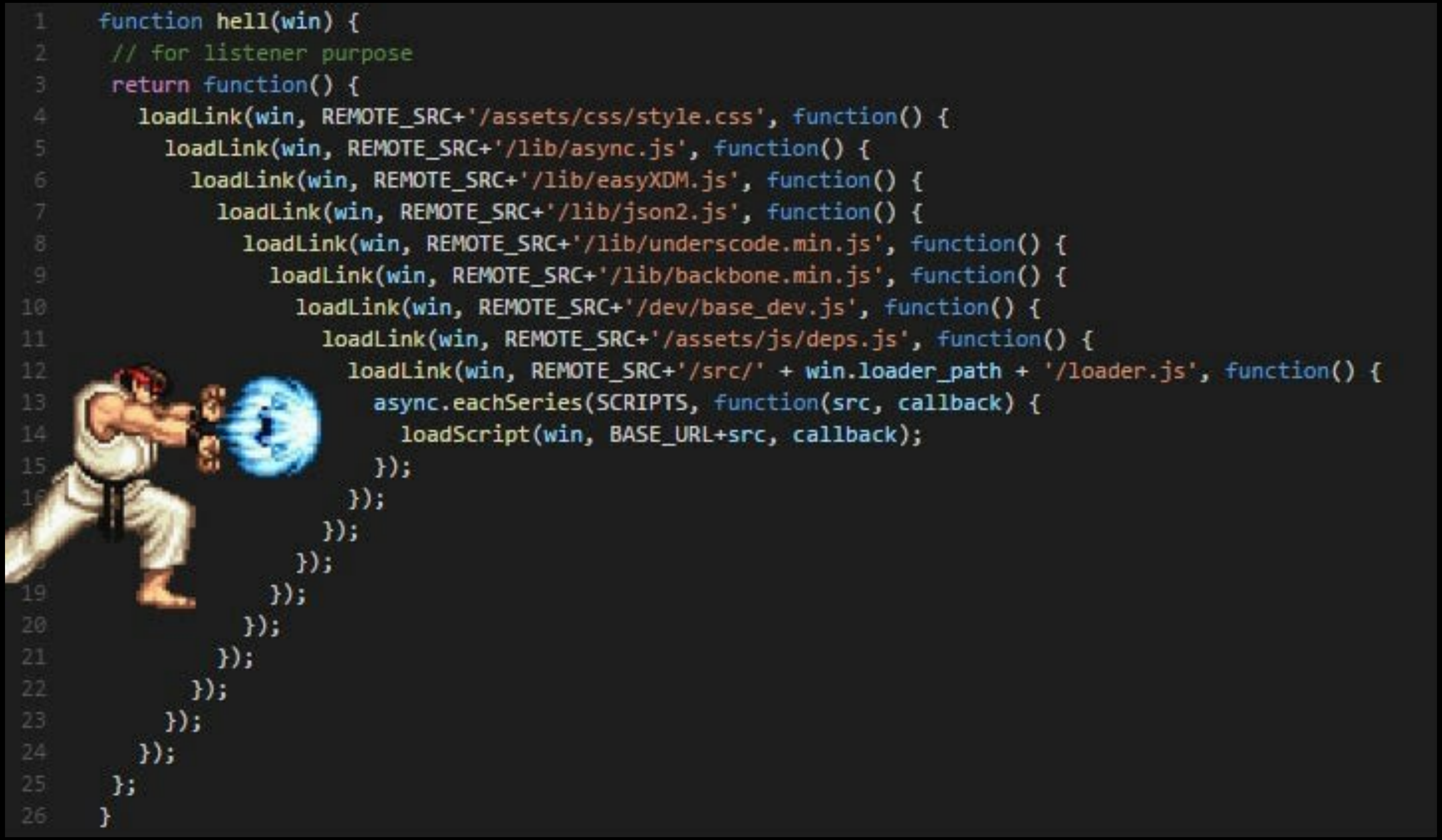
```
function sum(...numbers) {
    let total = 0;
    numbers.forEach(num => total += num);
    console.log(total);
    return total;
}
function sum(...numbers) {
    return numbers.reduce((total, num) => total + num);
}
console.log(sum(1, 2, 3, 4, 5, 6));
```

Callback

```
function loadImg(src, callback) {
    const img = new Image();
    img.onload = function () {
        callback && callback(img);
    };
    img.src = src;
}
```

```
function addImage(img) {
    document.body.appendChild(img);
    img.width = img.naturalWidth * 2;
    img.height = img.naturalHeight * 2;
}
loadImg("./background.jpg", addImage);
```

Callback Hell



```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Global Scope

Variables not inside any function is global scope

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Scope</title>
</head>
<body>
    <script src="./common.js"></script>
    <script src="./loadImage.js"></script>
</body>
</html>
```

```js
// common.js
const config = {
    designResolution: {
        width: 1280,
        height: 720,
    },
    orientation: "landscape"
}
// loadImage.js
const { designResolution } = config;
console.log(designResolution);
// {width: 1280, height: 720}
```

Local Scope

Variables inside function is local scope

```
function loadImg(src, callback) {
    const img = new Image();
    // local scope 1

    img.onload = function () {
        // local scope 2
        callback && callback(img);
    };
    img.src = src;
}
```

```
function loadImg(src, callback) {
    const img = new Image();
    console.log(scale); // error
    img.onload = function () {
        let scale = 2;
        callback && callback(img);
        img.width = img.naturalWidth * scale;
        img.height = img.naturalHeight * scale;
        console.log(img); // img
    };
    img.src = src;
}
```

Child scope including parent scope (lexical scope)

Block Scope

Write the function countdown time n(s) -> 0

```
function countDown(time) {
    for (var i = time; i >= 0; i--) {
        setTimeout(function () {
            console.log(i);
        }, 1000 * (time - i))
    }
}
countDown(3);
// -1 ... -1 ... -1
```

```
function countDown(time) {
  for (let i = time; i >= 0; i--) {
      setTimeout(function () {
          console.log(i);
      }, 1000 * (time - i))
  }
}
countDown(3);
// 3  ... 2 ... 1
```

let, const only work inside {}

var work inside the function

Scope

How they handle that before let, const

```
function countDown(time) {
    for (var i = time; i >= 0; i--) {
        (function(n){
            setTimeout(function () {
                console.log(n);
            }, 1000 * (time - i))
        })(i);
    }
}
```

```
function countDown(time) {
    for (var i = time; i >= 0; i--) {
        function delay(n){
            setTimeout(function () {
                console.log(n);
            }, 1000 * (time - i))
        }
        delay(i);
    }
}
```

Scope Summary

```
const { designResolution, isResize } = config;
// global scope
function addImage(img) {
   const { naturalWidth, naturalHeight } = img;
    // local scope of parent
    function getScale() {
        // local scope of child (lexical scope)
        const { width, height } = designResolution;
        return Math.min(naturalWidth / width, naturalHeight / height);
    }
    if (isResize) {
        // block scope
        const scale = getScale();
        img.width = naturalWidth * scale;
        img.height = naturalHeight * scale;
    }
}
```

Closures

```
Closures is all accessible variables when function created
```

```
const { designResolution, isResize } = config;
function addImage(img) {
    const { naturalWidth, naturalHeight } = img;
    function getScale() {
        const { width, height } = designResolution;
        return Math.min(naturalWidth / width, naturalHeight / height);
    }
    if (isResize) {
        const scale = getScale();
        return function () {
            // closure
            img.width = naturalWidth * scale;
            img.height = naturalHeight * scale;
        }
    }
}
```

Closures

Closures is all accessible variables when function created

- own scope
- parent scope
- global scope
- the arguments of the outer function
- the function has returned.

Context

What is this?

In an object method	this refers to the object.
Alone	this refers to the global object
In a function	this refers to the global object
In a function, in strict mode	this is undefined.
In an event	this refers to the element that received the event.

Context

What is this?

```
console.log(this);
// window
function log() {
    console.log(this);
    // "use strict" ? undefined : window
}
const dog = {
    bark: function(){
        console.log("gau gau", this);
        // {bark: f}
    }
}
```

```
window.onload = function (){

    const button = document.getElementById("Btn");

    button.addEventListener("click", dog.bark);

}

// gau gau <button id="BtnClick">Click Me</button>
```

call, apply

```
const dog = {
    sound: "gaugau",
    makeSound: function () {
        console.log(this.sound);
    }
}
const cat = {
    sound: "meomeo",
}
dog.makeSound.call(cat);
// meomeo
dog.makeSound.apply(cat);
// meomeo
```

```
const dog = {
    sound: "gaugau",
    makeSound: function (emo1, emo2) {
        console.log(this.sound, emo1, emo2);
    }
}
const cat = {
    sound: "meomeo",
}
dog.makeSound.call(cat, "hungry", "hungry");
// meomeo hungry hungry
dog.makeSound.aplly(cat, ["hungry","gruwwww"]);
// meomeo hungry gruwwww
```

bind

bind create a new function with a new this

```
const dog = {
    sound: "gaugau",
    makeSound: function (emo1, emo2) {
        console.log(this.sound, emo1, emo2);
    }
}
const cat = {
    sound: "meomeo",
}
cat.makeSound = dog.makeSound.bind(cat);
cat.makeSound("gruw","gruw");
// meomeo gruw gruw
```

arrow function

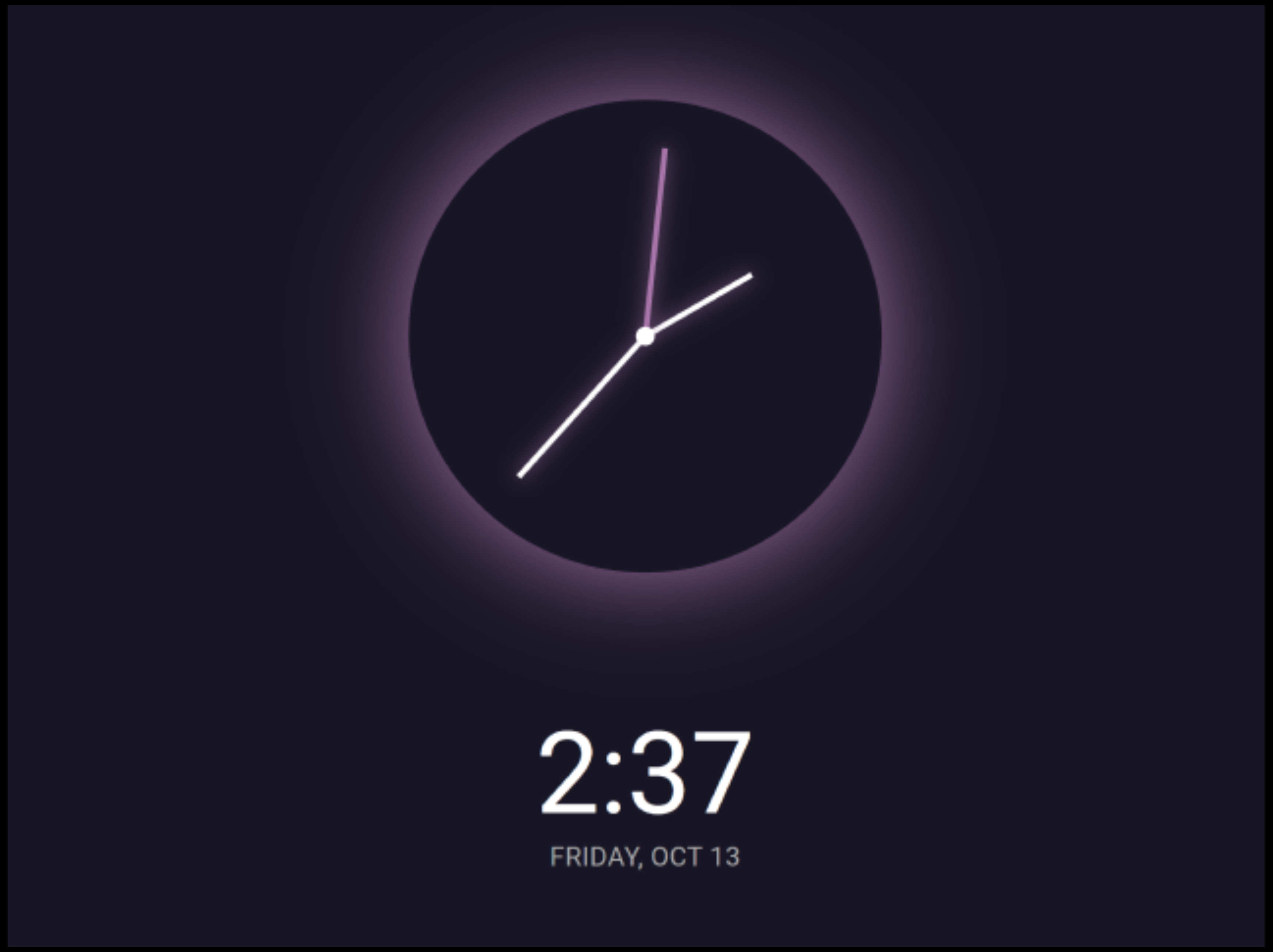
```
const max = function(a, b){
    return a > b ? a : b;
}
const max = (a, b) => a > b ? a : b;
```

- Do not have own this
- Do not have arguments
- Can not using with call, bind, apply
- Can not using as constructor
- Should not be used as object methods
- this is the current this when function created

Assignment 1

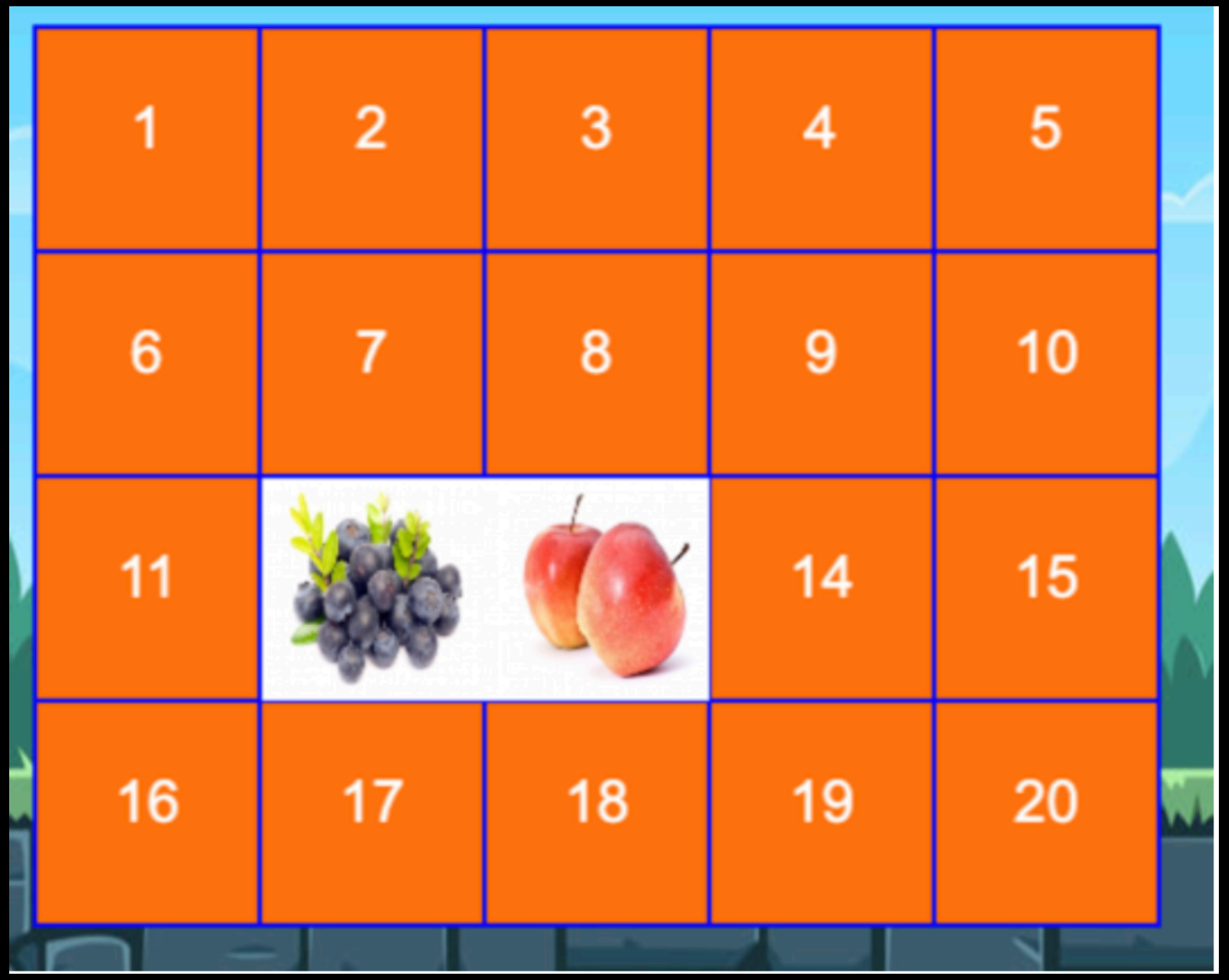
```
# Making a clock

- Create a clock like that image
```



Assignment 2

```
# The pair game
- There are 20 cards with 10 different images.
  - open 2 cards each time,
 - if they are matched, hide them, get 1000 coin.
 - if they are not, close them, loss 500 coin.
 - first coin is 10.000.
 - if coin < 0 => game over.
```



How to ask

```
1. What you want to do?

2. What did you do, and what is your issue?

3. Capture of your code or the error.
```

Homework