You Don't Know JS Yet: Get Started - 2nd Edition

Appendix B: Practice, Practice, Practice!

In this appendix, we'll explore some exercises and their suggested solutions. These are just to *get you started* with practice over the concepts from the book.

Practicing Comparisons

Let's practice working with value types and comparisons (Chapter 4, Pillar 3) where coercion will need to be involved.

`scheduleMeeting(..)` should take a start time (in 24hr format as a string "hh:mm") and a meeting duration (number of minutes). It should return `true` if the meeting falls entirely within the work day (according to the times specified in `dayStart` and `dayEnd`); return `false` if the meeting violates the work day bounds.

```
const dayStart = "07:30";
const dayEnd = "17:45";

function scheduleMeeting(startTime,durationMinutes) {
    // ..TODO..
}

scheduleMeeting("7:00",15);     // false
scheduleMeeting("07:15",30);    // false
scheduleMeeting("7:30",30);     // true
scheduleMeeting("11:30",60);    // true
scheduleMeeting("17:00",45);    // true
scheduleMeeting("17:30",30);    // false
scheduleMeeting("18:00",15);    // false
```

Try to solve this yourself first. Consider the usage of equality and relational comparison operators, and how coercion impacts this code. Once you have code that works, *compare* your solution(s) to the code in "Suggested Solutions" at the end of this appendix.

Practicing Closure

Now let's practice with closure (Chapter 4, Pillar 1).

The `range(..)` function takes a number as its first argument, representing the first number in a desired range of numbers. The second argument is also a number representing the end of the desired range (inclusive). If the second argument is omitted, then another function should be returned that expects that argument.

```
function range(start,end) {
    // ..TODO..
}

range(3,3);    // [3]
range(3,8);    // [3,4,5,6,7,8]
range(3,0);    // []

var start3 = range(3);
var start4 = range(4);

start3(3);     // [3]
start3(8);     // [3,4,5,6,7,8]
start3(0);     // []

start4(6);     // [4,5,6]
```

Try to solve this yourself first.

Once you have code that works, *compare* your solution(s) to the code in "Suggested Solutions" at the end of this appendix.

Practicing Prototypes

Finally, let's work on `this` and objects linked via prototype (Chapter 4, Pillar 2).

Define a slot machine with 3 reels that can individually `spin()`, and then `display()` the current contents of all the reels.

The basic behavior of a single reel is defined in the `reel` object below. But the slot machine needs individual reels -- objects that delegate to `reel`, and which each have a `position` property.

A reel only *knows how* to `display()` its current slot symbol, but a slot machine typically shows 3 symbols per reel: the current slot (`position`), one slot above (`position - 1`), and one slot below (`position + 1`). So displaying the slot machine should end up displaying a 3 x 3 grid of slot symbols.

```
function randMax(max) {
    return Math.trunc(1E9 * Math.random()) % max;
}

var reel = {
    symbols: [ "♠", "♥", "♦", "♣", "☺", "★", "☾", "☀" ],
    spin() {
        if (this.position == null) {
            this.position = randMax(this.symbols.length - 1);
        }
        this.position = (
            this.position + 100 + randMax(100)
        ) % this.symbols.length;
    },
    display() {
        if (this.position == null) {
```

```
            this.position = randMax(this.symbols.length - 1);
        }
        return this.symbols[this.position];
    }
};

var slotMachine = {
    reels: [
        // this slot machine needs 3 separate reels
        // hint: Object.create(..)
    ],
    spin() {
        this.reels.forEach(function spinReel(reel){
            reel.spin();
        });
    },
    display() {
        // TODO
    }
};

slotMachine.spin();
slotMachine.display();
// ☾ | ☀ | ★
// ☀ | ♠ | ☾
// ♠ | ♥ | ☀

slotMachine.spin();
slotMachine.display();
// ♦ | ♠ | ♣
// ♣ | ♥ | ☺
// ☺ | ♦ | ★
```

Try to solve this yourself first.

Hints:

1. use the `%` modulo operator for wrapping `position` as you access symbols circularly around a reel.
2. use `Object.create(..)` to create an object and prototype-link it to another object. Once linked, delegation allows the objects to share `this` context during method invocation.
3. instead of modifying the reel object directly to show each of the three positions, you can use another temporary object (`Object.create(..)` again) with its own `position` , to delegate from.

Once you have code that works, *compare* your solution(s) to the code in "Suggested Solutions" at the end of this appendix.

Suggested Solutions

Keep in mind that these suggested solutions are just that: suggestions. There's many different ways to solve these practice exercises. Compare your approach to what you see here, and consider the pros and cons of each.

Suggested solution for "Comparisons" (Pillar 3) practice:

```
const dayStart = "07:30";
const dayEnd = "17:45";

function scheduleMeeting(startTime,durationMinutes) {
    var [ , meetingStartHour, meetingStartMinutes ] =
        startTime.match(/^(\d{1,2}):(\d{2})$/) || [];

    durationMinutes = Number(durationMinutes);

    if (
        typeof meetingStartHour == "string" &&
        typeof meetingStartMinutes == "string"
    ) {
        let durationHours = Math.floor(durationMinutes / 60);
        durationMinutes = durationMinutes - (durationHours * 60);
        let meetingEndHour = Number(meetingStartHour) + durationHours;
        let meetingEndMinutes = Number(meetingStartMinutes) + durationMinutes;

        if (meetingEndMinutes > 60) {
            meetingEndHour = meetingEndHour + 1;
            meetingEndMinutes = meetingEndMinutes - 60;
        }

        // re-compose fully-qualified time strings
        // (to make comparison easier)
        let meetingStart = `${
            meetingStartHour.padStart(2,"0")
        }:${
            meetingStartMinutes.padStart(2,"0")
        }`;
        let meetingEnd = `${
            String(meetingEndHour).padStart(2,"0")
        }:${
            String(meetingEndMinutes).padStart(2,"0")
        }`;

        // NOTE: since expressions are all strings,
        // comparisons here are alphabetic, but that's
        // safe here since they're fully qualified
        // time strings (ie, "07:15" < "07:30")
        return (
            meetingStart >= dayStart &&
            meetingEnd <= dayEnd
        );
    }

    return false;
}

scheduleMeeting("7:00",15);     // false
scheduleMeeting("07:15",30);    // false
scheduleMeeting("7:30",30);     // true
scheduleMeeting("11:30",60);    // true
scheduleMeeting("17:00",45);    // true
```

```
scheduleMeeting("17:30",30);    // false
scheduleMeeting("18:00",15);    // false
```

Suggested solution for "Closure" (Pillar 1) practice:

```
function range(start,end) {
    start = Number(start) || 0;

    if (end === undefined) {
        return function getEnd(end) {
            return getRange(start,end);
        };
    }
    else {
        end = Number(end) || 0;
        return getRange(start,end);
    }


    // **********************

    function getRange(start,end) {
        var ret = [];
        for (let i = start; i <= end; i++) {
            ret.push(i);
        }
        return ret;
    }
}

range(3,3);    // [3]
range(3,8);    // [3,4,5,6,7,8]
range(3,0);    // []

var start3 = range(3);
var start4 = range(4);

start3(3);     // [3]
start3(8);     // [3,4,5,6,7,8]
start3(0);     // []

start4(6);     // [4,5,6]
```

Suggested solution for "Prototypes" (Pillar 2) practice:

```
function randMax(max) {
    return Math.trunc(1E9 * Math.random()) % max;
}

var reel = {
    symbols: [ "♠", "♥", "♦", "♣", "☺", "★", "☾", "☀" ],
    spin() {
        if (this.position == null) {
            this.position = randMax(this.symbols.length - 1);
```

```
        }
        this.position = (
            this.position + 100 + randMax(100)
        ) % this.symbols.length;
    },
    display() {
        if (this.position == null) {
            this.position = randMax(this.symbols.length - 1);
        }
        return this.symbols[this.position];
    }
};

var slotMachine = {
    reels: [
        Object.create(reel),
        Object.create(reel),
        Object.create(reel)
    ],
    spin() {
        this.reels.forEach(function spinReel(reel){
            reel.spin();
        });
    },
    display() {
        var lines = [];

        // display all 3 lines on the slot machine
        for (let linePos = -1; linePos <= 1; linePos++) {
            let line = this.reels.map(function getSlot(reel){
                var slot = Object.create(reel);
                slot.position = (
                    reel.symbols.length + reel.position + linePos
                ) % reel.symbols.length;
                return reel.display.call(slot);
            });
            lines.push(line.join(" | "));
        }

        return lines.join("\n");
    }
};

slotMachine.spin();
slotMachine.display();
// ☾ | ☀ | ★
// ☀ | ♠ | ☾
// ♠ | ♥ | ☀

slotMachine.spin();
slotMachine.display();
// ♦ | ♠ | ♣
// ♣ | ♥ | ☺
// ☺ | ♦ | ★
```

That's it for this book. But now it's time to look for real projects to practice these ideas on. Just keep coding, because that's the best way to learn!