Problems

Douglas Crockford

```
function funky(o) {
    o = null;
}
var x = [];
funky(x);
alert(x);
```

A. null
B. []
C. undefined
D. throw

What is x?

```
function swap(a, b) {
    var temp = a;
    a = b;
    b = temp
}
var x = 1, y = 2;
swap(x, y);
alert(x);
```

A. 1
B. 2
C. undefined
D. throw

What is x?

Write a function that takes an argument returns that argument.

```
identity(3)    // 3
```

```
function identity(x) {
    return x;
}
```

```
var identity = function identity(x) {
    return x;
};
```

Write two binary functions, **`add`** and **`mul`**, that take two numbers and return their sum and product.

```
add(3, 4)     //  7
mul(3, 4)     // 12
```

```
function add(x, y) {
    return x + y;
}

function mul(x, y) {
    return x * y;
}
```

Write a function that takes an argument and returns a function that returns that argument.

```
idf = identityf(3);

idf()    // 3
```

```
function identityf(x) {
    return function () {
        return x;
    };
}
```

Write a function that adds from two invocations.

```
addf(3)(4)     // 7
```

```
function addf(x) {
    return function (y) {
        return x + y;
    };
}
```

Write a function that takes a binary function, and makes it callable with two invocations.

```
addf = applyf(add);
addf(3)(4)                  // 7
applyf(mul)(5)(6)           // 30
```

```
function applyf(binary) {
    return function (x) {
        return function (y) {
            return binary(x, y);
        };
    };
}
```

Write a function that takes a function and an argument, and returns a function that can supply a second argument.

```
add3 = curry(add, 3);
add3(4)                    // 7

curry(mul, 5)(6)           // 30
```

```
function curry(func, first) {
    return function (second) {
        return func(first, second);
    };
}
```

```
function curry(func, first) {
    return function (second) {
        return func(first, second);
    };
}
```

currying

schönfinkelisation

```
function curry(func, first) {
    return function (second) {
        return func(first, second);
    };
}

function curry(func, first) {
    return applyf(func)(first);
}
```

```
function curry(func) {
    var slice = Array.prototype.slice,
        args = slice.call(arguments, 1);
    return function () {
        return func.apply(
           null,
           args.concat(slice.call(arguments,
0))
        );
    };
}
```

```
function curry(func) {
    var slice = Array.prototype.slice,
        args = slice.call(arguments, 1);
    return function () {
        return func.apply(
           null,
           args.concat(slice.call(arguments,
0))
        );
    };
}

function curry(func, ...first) {
    return function (...second) {
        return func(...first, ...second);
    };
}
```

Without writing any new functions, show three ways to create the `inc` function.

```
inc(5)                  // 6
inc(inc(5))             // 7
```

```
1. inc = addf(1);

2. inc = applyf(add)(1);

3. inc = curry(add, 1);
```

Write **`methodize`**, a function that converts a binary function to a method.

```
Number.prototype.add =
    methodize(add);
(3).add(4)     // 7
```

```
function methodize(func) {
    return function (y) {
        return func(this, y);
    };
}

function methodize(func) {
    return function (...y) {
        return func(this, ...y);
    };
}
```

Write **`demethodize`**, a function that converts a method to a binary function.

```
demethodize(Number.prototype.add)(5, 6)
    // 11
```

```
function demethodize(func) {
    return function (that, y) {
        return func.call(that, y);
    };
}

function demethodize(func) {
    return function (that, ...y) {
        return func.apply(that, y);
    };
}
```

Write a function **`twice`** that takes a binary function and returns a unary function that passes its argument to the binary function twice.

```
var double = twice(add);
double(11)     // 22
var square = twice(mul);
square(11)     // 121
```

```
function twice(binary) {
    return function (a) {
        return binary(a, a);
    };
}
```

Write a function **`composeu`** that takes two unary functions and returns a unary function that calls them both.

```
composeu(double, square)(3)    // 36
```

```
function composeu(f, g) {
    return function (a) {
        return g(f(a));
    };
}
```

Write a function **`composeb`** that takes two binary functions and returns a function that calls them both.

```
composeb(add, mul)(2, 3, 5)     // 25
```

```
function composeb(f, g) {
    return function (a, b, c) {
        return g(f(a, b), c);
    };
}
```

Write a function that allows another function to only be called once.

```
add_once = once(add);
add_once(3, 4)      // 7
add_once(3, 4)      // throw!
```

```
function once(func) {
    return function () {
        var f = func;
        func = null;
        return f.apply(
            this,
            arguments
        );
    };
}
```

Write a factory function that returns two functions that implement an up/down counter.

```
counter = counterf(10);
counter.inc()     // 11
counter.dec()     // 10
```

```
function counterf(value) {
    return {
        inc: function () {
            value += 1;
            return value;
        },
        dec: function () {
            value -= 1;
            return value;
        }
    };
}
```

Make a revocable function that takes a nice function, and returns a **`revoke`** function that denies access to the nice function, and an **`invoke`** function that can invoke the nice function until it is revoked.

```
temp = revocable(alert);
temp.invoke(7);     // alert: 7
temp.revoke();
temp.invoke(8);     // throw!
```

```
function revocable(nice) {
    return {
        invoke: function () {
            return nice.apply(
                this,
                arguments
            );
        },
        revoke: function () {
            nice = null;
        }
    };
}
```

Make an array wrapper object with methods **`get`**, **`store`**, and **`append`**, such that an attacker cannot get access to the private array.

```
my_vector = vector();
my_vector.append(7);
my_vector.store(1, 8);
my_vector.get(0)     // 7
my_vector.get(1)     // 8
```

```
function vector() {
    var array = [];

    return {
        get: function (i) {
            return array[i];
        },
        store: function store(i, v) {
            array[i] = v;
        },
        append: function (v) {
            array.push(v);
        }
    };
}
```

```
function vector() {
    var array = [];

    return {
        get: function (i) {
            return array[i];
        },
        store: function store(i, v) {
            array[i] = v;
        },
        append: function (v) {
            array.push(v);
        }
    };
}
```

```
var stash;
table.store('push', function () {
    stash = this;
});
table.append(); // stash === array
```

```
function vector() {
    var array = [];

    return {
        get: function (i) {
            return array[+i];
        },
        store: function store(i, v) {
            array[+i] = v;
        },
        append: function (v) {
            array[array.length] = v;
        }
    };
}
```

Make a function that makes a publish/subscribe object. It will reliably deliver all publications to all subscribers in the right order.

```
my_pubsub = pubsub();
my_pubsub.subscribe(alert);
my_pubsub.publish("It works!");
    // alert("It works!")
```

```
function pubsub() {
    var subscribers = [];
    return {
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                subscribers[i](publication);
            }
        }
    };
}
```

```
function pubsub() {
    var subscribers = [];
    return {
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                subscribers[i](publication);
            }
        }
    };
}
```

```
my_pubsub.subscribe = function () {
};
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                subscribers[i](publication);
            }
        }
    });
}
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                subscribers[i](publication);
            }
        }
    });
}
```

```
my_pubsub.subscribe(function () {
    throw "None for the rest";
});
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                try {
                    subscribers[i](publication);
                } catch (ignore) {}
            }
        }
    });
}
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            var i, length = subscribers.length;
            for (i = 0; i < length; i += 1) {
                try {
                    subscribers[i](publication);
                } catch (ignore) {}
            }
        }
    });
}
```

```
my_pubsub.subscribe(function () {
    this.length = 0;
});
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            subscribers.forEach(function (s) {
                try {
                    s(publication);
                } catch (ignore) {}
            });
        }
    });
}
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            subscribers.forEach(function (s) {
                try {
                    s(publication);
                } catch (ignore) {}
            });
        }
    });
}
```

```
my_pubsub.subscribe(once(function () {
    my_pubsub.publish("Out of order");
}));
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: setTimeout(function (publication) {
            subscribers.forEach(function (s) {
                try {
                    s(publication);
                } catch (ignore) {}
            });
        }, 0)
    });
}
```

```
function pubsub() {
    var subscribers = [];
    return Object.freeze({
        subscribe: function (subscriber) {
            subscribers.push(subscriber);
        },
        publish: function (publication) {
            subscribers.forEach(function (s) {
                setTimeout(function () {
                    s(publication);
                }, 0);
            });
        }
    });
}
```

Make a factory that makes functions that generate unique symbols.

```
gensym = gensymf('G');
gensym()      // 'G0'
gensym()      // 'G1'
gensym()      // 'G2'
gensym()      // 'G3'
```

```
function gensymf(prefix) {
    var number = 0;
    return function () {
        number += 1;
        return prefix + number;
    };
}
```

Make a function that returns a function that will return the next fibonacci number.

```
var fib = fibonaccif(0, 1);
fib()     // 0
fib()     // 1
fib()     // 1
fib()     // 2
fib()     // 3
fib()     // 5
```

```
function fibonaccif(a, b) {
    return function () {
        var next = a;
        a = b;
        b += next;
        return next;
    };
}
```

Write a function that adds from many invocations, until it sees an empty invocation.

```
addg(3)(4)(5)()          // 12
addg(1)(2)(4)(8)()       // 15
```

```
function addg(x) {
    return function step(y) {
        if (y === undefined) {
            return x;
        }
        x += y;
        return step;
    };
}
```

Write a function that will take a binary function and apply it to many invocations.

```
applyg(add)(3)(4)(5)()      // 12
applyg(add)(1)(2)(4)(8)()   // 15
```

```
function applyg(binary) {
    var x;
    return function step(y) {
        if (y === undefined) {
            return x;
        }
        x = x === undefined
            ? y
            : binary(x, y);
        return step;
    };
}
```

Write a function `m` that takes a value and an optional source string and returns them in an object.

```
JSON.stringify(m(1))

// {"value": 1, "source": "1"}

JSON.stringify(m(Math.PI, "pi"))

// {"value": 3.14159…, "source":
"pi"}
```

```
function m(value, source) {
    return {
        value: value,
        source: value ===
            undefined
            ? String(value)
            : source
    };
}
```

Write a function **`addm`** that takes two **`m`** objects and returns an **`m`** object.

```
JSON.stringify(addm(m(3), m(4)))
// {"value": 3, "source": "(3+4)"}
```

```
function addm(a, b) {
    return m(
        a.value + b.value,
        "(" + a.source + "+" +
            b.source + ")"
    );
}
```

Write a function **`binarymf`** that takes a binary function and a string and returns a function that acts on **`m`** objects.

```
addm = binarymf(add, "+");

JSON.stringify(addm(m(3), m(4)))

// {"value": 7, "source": "(3+4)"}
```

```
function binarymf(f, op) {
    return function (a, b) {
        return m(
            f(a.value, b.value),
            "(" + a.source + op +
                b.source + ")"
        );
    };
}
```

Modify function **`binarymf`** so that the functions it produces can accept arguments that are either numbers or **`m`** objects.

```
addm = binarymf(add, "+");
JSON.stringify(addm(3, 4))
// {"value": 7, "source": "(3+4)"}
```

```
function binarymf(f, op) {
    return function (a, b) {
        if (typeof a === 'number') {
            a = m(a);
        }
        if (typeof b === 'number') {
            b = m(b);
        }
        return m(
            f(a.value, b.value),
            "(" + a.source + op +
                b.source + ")"
        );
    };
}
```

Write function **`unarymf`**, which is like **`binarymf`** except that it acts on unary functions.

```
squarem = unarymf(square, "square");
JSON.stringify(squarem(4))
// {"value": 16,
    "source": "(square 4)"}
```

```
function unarymf(f, op) {
    return function (a) {
        if (typeof a === 'number') {
            a = m(a);
        }
        return m(
            f(a.value),
            "(" + op + " " +
                a.source + ")"
        );
    };
}
```

Write a function that takes the lengths of two sides of a triangle and computes the length of the hypotenuse. (Hint: $c^2 = a^2 + b^2$)

```
hyp(3, 4)     // 5
```

```
function hyp(a, b) {
    return Math.sqrt(
        add(
            mul(a, a),
            mul(b, b)
        )
    );
}
```

Write a function that evaluates array expressions.

```
hypa = [
     Math.sqrt,
      [
           add,
            [mul, 3, 3],
            [mul, 4, 4]
      ]
];
exp(hypa)     // 5
```

```
function exp(value) {
    return Array.isArray(value)
        ? value[0](
            exp(value[1]),
            exp(value[2])
        )
        : value;
}
```

Make a function that stores a value in a variable.

```
var variable;
store(5);    // variable === 5
```

```
var variable;
function store(value) {
    variable = value;
}
```

Make a function that takes a binary function, two functions that provide operands, and a function that takes the result.

```
quatre(
    add,
    identityf(3),
    identityf(4),
    store
);      // variable === 7
```

```
function quatre(
    binary,
    fx,
    fy,
    fresult
) {
    return fresult(
        binary(fx(), fy())
    );
}
```

Make a function that takes a unary function, and returns a function that takes an argument and a callback.

```
sqrtc = unaryc(Math.sqrt);
sqrt(81, store) // variable === 9
```

```
function unaryc(unary) {
    return function (x, c) {
        return c(unary(x));
    };
}
```

Make a function that takes a binary function, and returns a function that takes two arguments and a callback.

```
addc = binaryc(add);
addc(4, 5, store) // variable === 9
mulc = binaryc(mul);
mulc(2, 3, store) // variable === 6
```

```
function binaryc(unary) {
    return function (x, y, c) {
        return c(binary(x, y));
    };
}
```

Write the **`hypc`** (hyp with continuation) function using addc, mulc, and sqrtc.

```
hypc(3, 4, alert)  // alert: 5
```

```
function hypc(a, b, c) {
  return mulc(a, a, function (a2) {
    return mulc(b, b, function (b2) {
      return addc(a2, b2, function (a2b2) {
        return sqrtc(a2b2, c);
      });
    });
  });
}
```

Take a break.