JavaScript the Good Parts

By Douglas Crockford

**Problem 1**

**function funky(o) {**

**o = null;**

**}**

**var x = [];**

**funky(x);**

**alert(x);**

**What is x?**

1. **null**
2. **[]**
3. **C.undefined**
4. **D. throw**

Ans : B. [ ]

**Problem 2**

**function swap(a,b) {**

**var temp = a;**

**a = b;**

**b = temp;**

**}**

**var x = 1, y = 2;**

**swap(x,y);**

**alert(x);**

**What is x?**

1. **1**
2. **2**
3. **undefined**
4. **throw**

Ans : A. 1

**Problem 3: Write a function that takes an argument and returns that argument.**

**function returnValue(n){**

**return n;**

**}**

**alert(returnValue(3)); //3**

**Problem 4: Write two binary functions, and and mul, that take two numbers and return their sum and product.**

**function add(a, b){**

**return (a+b);**

**}**

**function mul(a, b){**

**return a\*b;**

**}**

**alert(add(3,4)); //7**

**alert(mul(3,4)); //12**

**Problem 5: Write a function that takes an argument and returns a function that returns that argument.**

**function returnValuef(n){**

**return function() {**

**return n;**

**}**

**};**

**Idf = returnValuef(3);**

**Idf(); //3**

**Problem 6: Write a function that adds from two invocations.**

**function addf(x){**

**return function (y) {**

**return x + y;**

**}**

**};**

**addf(3)(4); //7**

**Problem 7: Write a function that takes a binary function and takes a binary function, and make it callable with two invocations.**

**function applyf(func){**

**return function (x){**

**return function (y) {**

**return func(x,y);**

**};**

**};**

**}**

**addf = applyf(add);**

**addf(3)(4); //7**

**applyf(mul)(5)(6) //30**

**Problem 8: Write a function that takes a function and an argument, and returns a function that can supply a second argument.**

**function curry(func, x){**

**return function (y) {**

**return func(x,y);**

**};**

**}**

**Add3 = curry(add,3);**

**Add3(4); //7**

**curry(mul,5)(6); //30**

**// extra credit**

**function curry(func,x){**

**return applyf(func)(x);**

**}**

**Problem 9: Without writing any new functions, show three ways to create the inc function.**

**inc(5) //6**

**inc(inc(6)) //7**

1. **inc = addf(1);**
2. **inc = applyf(add)(1);**
3. **inc = curry(add,1);**

**Problem 10: 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);**

**};**

**}**

**// for multiple arg**

**function methodize(func){**

**return function(...y){**

**return func(this,...y);**

**};**

**}**

**Problem 11: Write demethodize, a function that converts a method to a binary function.**

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

**function demethodize(func){**

**return function(x,y){**

**return func(x,y);**

**};**

**}**

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

**// for multiple arg**

**function demethodize(func){**

**return function(x,...y){**

**return func(x,y);**

**};**

**}**

**Problem 12: Write a function twice that takes a binary function and returns a unary function that passes its argument to the binary function twice.**

**function twice(func){**

**return function(x){**

**return func(x,x);**

**};**

**}**

**var double = twice(add);**

**double(11) //22**

**var square = twice(mul);**

**square(11) //121**

**Problem 13: Write a function compseu that takes two unary functions and returns a unary function that calls both of them.**

**function composeu(f, g){**

**return function(x){**

**return g(f(x));**

**};**

**}**

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

**Problem 14: Write a function compseb that takes two binary functions and returns a function that calls both of them.**

**function composeb(f,g){**

**return function(x,y,z){**

**return g(f(x,y),z);**

**};**

**}**

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

**Problem 15: Write a function that allows another function to only be called once.**

**function once(func){**

**return function(){**

**var f = func;**

**func = null;**

**return f.apply(this,arguments);**

**};**

**}**

**var add\_once = once(add);**

**add\_once(3,4); //7**

**add\_once(3,4); //throw, because now add function is null**

**Problem 16: Write a factory function that returns two functions that implement an up/down counter.**

**function counterf(x){**

**return {**

**inc : function (){**

**x += 1;**

**return x;**

**},**

**dec : function (){**

**x -= 1;**

**return x - 1;**

**},**

**};**

**}**

**counter = counterf(10);**

**counter.inc() //11**

**counter.dec() //10**

**Problem 17: 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’s revoked.**

**function revocable(nice){**

**return {**

**invoke : function(){**

**return nice.apply(**

**this,**

**arguments**

**);**

**},**

**revoke : function(){**

**nice = null;**

**},**

**};**

**}**

**temp = revocable(alert);**

**temp.invoke(7); //alert : 7**

**temp.revoke();**

**temp.invoke(8); //throw!**

**// another way**

**function revocable(nice){**

**return {**

**invoke : function(value){**

**nice(value);**

**},**

**revoke : function(){**

**nice = null;**

**},**

**};**

**}**

**Problem 18: Write a limit function that allows a binary function to be called a limited number of times.**

**function limit(func,lim){**

**return function(x,y){**

**if(lim > 0){**

**lim -= 1;**

**return add(x,y);**

**}**

**return undefined;**

**};**

**}**

**var add\_ltd = limit(add,1);**

**add\_ltd(3,4) //7**

**add\_ltd(3,5) //undefined**

**Problem 19: Write a from function that produces a generator that will produce a series of values.**

**function from(x){**

**return function(){**

**return x++;**

**};**

**}**

**var index = from(0);**

**index() // 0**

**index() // 1**

**index() // 2**

**Problem 19: Write a ‘to’ function that takes a generator and an end value, and returns a generator that will produce numbers up to that limit.**

**function to(func, end){**

**return function(){**

**var value = func();**

**if(value < end){**

**return value;**

**}**

**return undefined;**

**};**

**}**

**var index = to(from(1), 3);**

**index() // 1**

**index() // 2**

**index() // undefined**

**Problem 20: Write a fromTo function that produces a generator that will produce values in a range.**

**function fromTo(start, end){**

**return function(){**

**if(start < end){**

**start++;**

**return start;**

**}**

**return undefined;**

**};**

**}**

**// extra**

**function fromTo(start, end){**

**return to(**

**from(start),**

**end**

**);**

**}**

**var index = fromTo(0, 3);**

**index() // 0**

**index() // 1**

**index() // 2**

**index() // undefined**

**Problem 21: Write an element function that takes an array and a generator and returns a generator that will produce elements from the array.**

**function element(arr, gen){**

**return function(){**

**var index= gen();**

**if(index !== undefined){**

**return arr[index];**

**}**

**};**

**}**

**var ele = element([**

**'a', 'b', 'c', 'd'**

**], fromTo(1, 3));**

**ele() // 'b'**

**ele() // 'c'**

**ele() // undefined**

**Problem 22: Modify the element function so that the generator argument is optional. If a generator is not provided, then each of the elements of the array will be produced.**

**function element(arr, gen){**

**if(gen === undefined){**

**gen = fromTo(0, arr.length);**

**}**

**return function(){**

**var index= gen();**

**if(index !== undefined){**

**return arr[index];**

**}**

**};**

**}**

**var ele = element([**

**'a', 'b', 'c', 'd'**

**]);**

**ele() // 'a'**

**ele() // 'b'**

**ele() // 'c'**

**ele() // 'd'**

**ele() // undefined**

**Problem 23: Write a collect function that takes a generator and an array and produces a function that will collect the results in the array.**

**function collect(gen, arr){**

**return function(){**

**var value = gen();**

**if(value !== undefined){**

**arr.push(value);**

**}**

**return value;**

**};**

**}**

**var array = [],**

**col = collect(fromTo(0,2),array);**

**col() //0**

**col() //1**

**col() //undefined**

**array //[0,1]**

**Problem 24: Write a filter function that takes a generator and a predicate and produces a generator that produces only the values approved by the predicate.**

**function filter(gen, pred){**

**return function(){**

**var value = gen();**

**while(value !== undefined && !pred(value)){**

**value = gen();**

**}**

**return value;**

**};**

**}**

**var fil = filter(fromTo(0, 5),**

**function third(value) {**

**return (value % 3) === 0;**

**});**

**fil() // 0**

**fil() // 3**

**fil() // undefined**

**Problem 25: Write a concat function that takes two generators and produces a generator that combines the sequences.**

**function concat(f,g){**

**return function(){**

**var value = f();**

**if(value === undefined){**

**value = g();**

**}**

**return value;**

**};**

**}**

**var con = concat(fromTo(0, 3),**

**fromTo(0,2));**

**con() // 0**

**con() // 1**

**con() // 2**

**con() // 0**

**con() // 1**

**con() // undefined**

**Problem 26: Write a repeat function that takes a generator and calls it until it returns undefined.**

**function repeat(gen){**

**if(gen() !== undefined){**

**return repeat(gen);**

**}**

**}**

**var array = [];**

**repeat(collect(fromTo(0, 4), array));**

**log(array); // 0, 1, 2, 3**

**Problem 27: Write a map function that takes an array and a unary function, and returns an array containing the result of passing each element to the unary function. Use the repeat function.**

**function map(arr, unary){**

**var index = 0;**

**while(index < arr.length){**

**arr[index] = unary(arr[index]);**

**index += 1;**

**}**

**return arr;**

**}**

**// extra credit**

**function map(arr,unary){**

**var ele = element(array), result = [];**

**repeat(collect(function (){**

**var value = ele();**

**if(value !== undefined){**

**return unary(value);**

**}**

**},result));**

**return result;**

**}**

**map([2, 1, 0], inc) // [3, 2, 1]**

**Problem 28: Write a reduce function that takes an array and a binary function, and returns a single value.   
Use the repeat function.**

**function reduce(arr, binary){**

**var ele = element(array),result;**

**repeat(function(){**

**var value = ele();**

**if(value !== undefined){**

**if(result === undefined){**

**result = value;**

**}else{**

**result = binary(result,value);**

**}**

**}**

**return value;**

**});**

**return result;**

**}**

**reduce([], add) // undefined**

**reduce([2], add) // 2**

**reduce([2, 1, 0], add) // 3**

**Problem 29: Make a function gensymf that makes a function that generates unique symbols.**

**function gensymf(sym){**

**var number = from(1);**

**return function(){**

**return sym + number();**

**};**

**}**

**var geng = gensymf("G"),**

**genh = gensymf("H");**

**geng() // "G1"**

**genh() // "H1"**

**geng() // "G2"**

**genh() // "H2"**

**Problem 30: Write a function gensymff that takes a unary function and a seed and returns a gensymf.**

**function gensymff(unary,seed){**

**return function(char){**

**var number = seed;**

**return function(){**

**number = unary(number);**

**return char + number;**

**};**

**};**

**}**

**var gensymf = gensymff(inc, 0),**

**geng = gensymf("G"),**

**genh = gensymf("H");**

**geng() // "G1"**

**genh() // "H1"**

**geng() // "G2"**

**genh() // "H2"**

**Problem 31: Make a function fibonaccif that returns a generator that will return the next fibonacci number.**

**function fibonaccif(first,second){**

**var i = 0;**

**return function(){**

**var third;**

**switch(i){**

**case 0:**

**i = 1;**

**return first;**

**case 1:**

**i = 2;**

**return second;**

**default:**

**third = first + second;**

**first = second;**

**second = third;**

**return third;**

**}**

**};**

**}**

**// extra credit**

**function fibonaccif(first,second){**

**return function(){**

**var next = first;**

**first = second;**

**second += next;**

**return next;**

**};**

**}**

**//**

**function fibonaccif(a, b) {**

**return concat(**

**concat(**

**limit(identityf(a), 1),**

**limit(identityf(b), 1)**

**), function fibonacci() {**

**var next = a + b;**

**a = b;**

**b = next;**

**return next;**

**}**

**);**

**}**

**//**

**function fibonaccif(a, b) {**

**return concat(**

**element([a, b]),**

**function fibonacci() {**

**var next = a + b;**

**a = b;**

**b = next;**

**return next;**

**}**

**);**

**}**

**var fib = fibonaccif(0, 1);**

**fib() // 0**

**fib() // 1**

**fib() // 1**

**fib() // 2**

**fib() // 3**

**fib() // 5**

**Problem 32: Write a counter function that returns an object containing two functions that implement an up/down counter, hiding the counter.**

**function counter(num){**

**return {**

**up : return function(){**

**num += 1;**

**return num;**

**},**

**down:return function(){**

**num -= 1;**

**return num;**

**},**

**};**

**}**

**var object = counter(10),**

**up = object.up,**

**down = object.down;**

**up() // 11**

**down() // 10**

**down() // 9**

**up() // 10**

**Problem 32: Write a function m that takes a value and an optional source string and returns them in an object.**

**function m(value,str){**

**if(str === undefined){**

**str = value.toString();**

**}**

**return {**

**"value" : value,**

**"source": str,**

**};**

**}**

**JSON.stringify(m(1))**

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

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

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

**Problem 33: Write a function addm that takes two m objects and returns an m object.**

**function addm(first,second){**

**return {**

**"value" : first["value"] + second["value"],**

**"source": "(" +**

**first["source"] +**

**"+" +**

**second["source"] +**

**")",**

**};**

**}**

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

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

**JSON.stringify(addm(m(1), m(Math.PI, "pi")))**

**// {"value": 4.14159…, "source": "(1+pi)"}**

**//extra credit**

**function addm(first,second){**

**return m(**

**first.value + second.value,**

**"(" + first.source + "+" + second.source + ")"**

**);**

**}**

**Problem 33: Write a function liftm that takes a binary function and a string and returns a function that acts on m objects.**

**function liftm(binary, str){**

**return function(first,second){**

**return m(**

**binary(first.value, second.value),**

**"(" + first.source + str + second.source + ")"**

**);**

**};**

**}**

**var addm = liftm(add, "+");**

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

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

**JSON.stringify(liftm(mul, "\*")(m(3),**

**m(4)))**

**// {"value": 12, "source": "(3\*4)"}**

**Problem 34: Modify function liftm so that the functions it produces can accept arguments that are either numbers or m objects.**

**function liftm(binary, str){**

**return function(first,second){**

**if(typeOf(first) === "number"){**

**first = m(first);**

**}**

**if(typeOf(second) === "number"){**

**second= m(second);**

**}**

**return m(**

**binary(first.value, second.value),**

**"(" + first.source + str + second.source + ")"**

**);**

**};**

**}**

**var addm = liftm(add, "+");**

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

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

**Problem 35: Write a function exp that evaluates simple array expressions.**

**function exp(arr){**

**return (Array.isArray(arr))? value[0](value[1],value[2]):arr;**

**}**

**var sae = [mul, 5, 11];**

**exp(sae) // 55**

**exp(42) // 42**

**Problem 36: Modify exp to evaluate nested array expressions.**

**function exp(arr){**

**return (Array.isArray(arr))**

**? value[0](**

**exp(value[1]),**

**exp(value[2])**

**)**

**:arr;**

**}**

**var nae = [**

**Math.sqrt,**

**[**

**add,**

**[square, 3],**

**[square, 4]**

**]**

**];**

**exp(nae) // 5**

**Problem 37: Write a function addg that adds from many invocations, until it sees an empty invocation.**

**function addg(first){**

**if(first === undefined)**

**return;**

**return function(second){**

**if(second !== undefined)**

**return addg(first+second);**

**return first;**

**};**

**}**

**addg() // undefined**

**addg(2)() // 2**

**addg(2)(7)() // 9**

**addg(3)(0)(4)() // 7**

**addg(1)(2)(4)(8)() // 15**

**Problem 38: Write a function liftg that will take a binary function and apply it to many invocations.**

**function liftg(binary){**

**return function(second){**

**if(second === undefined){**

**return second;**

**}**

**return function more(next){**

**if(next === undefined){**

**return second;**

**}**

**second = binary(second,next);**

**return more;**

**};**

**};**

**}**

**liftg(mul)() // undefined**

**liftg(mul)(3)() // 3**

**liftg(mul)(3)(0)(4)() // 0**

**liftg(mul)(1)(2)(4)(8)() // 64**

**Problem 39: Write a function arrayg that will build an array from many invocations.**

**function arrayg(first){**

**var arr = [];**

**function more(second){**

**if(second !== undefined){**

**arr.push(second);**

**return more;**

**}**

**return arr;**

**}**

**return more(first);**

**}**

**// extra**

**function arrayg(first) {**

**if (first === undefined) {**

**return [];**

**}**

**return liftg(**

**function (array, value) {**

**array.push(value);**

**return array;**

**}**

**)([first]);**

**}**

**arrayg() // []**

**arrayg(3)() // [3]**

**arrayg(3)(4)(5)() // [3, 4, 5]**

**Problem 40: Make an array wrapper object with methods get, store, and append, such that an attacker cannot get access to the private array.**

**function vector(){**

**var arr = [];**

**return {**

**append : function(num){**

**arr.push(num);**

**},**

**store : function(index,num){**

**arr[1] = num;**

**},**

**get : function(index){**

**return arr[index];**

**},**

**};**

**}**

**myvector = vector();**

**myvector.append(7);**

**myvector.store(1, 8);**

**myvector.get(0) // 7**

**myvector.get(1) // 8**

**Problem 41: Make a function that makes a publish/subscribe object. It will reliably deliver all publications to all subscribers in the right order.**

**function pubsub(){**

**var sub= [];**

**return {**

**subscribe : function(func){**

**if(func !== undefined){**

**sub.push(func);**

**}**

**},**

**publish : function(pubStr){**

**for(var i=0;i<sub.length;i++){**

**sub[i](pubStr);**

**}**

**},**

**};**

**}**

**my\_pubsub = pubsub();**

**my\_pubsub.subscribe(log);**

**my\_pubsub.publish("It works!");**

**// log("It works!")**