**JavaScript The Good Parts**

by Douglas Crockford

**Problem-1**

function funky(o) {

o = null ;

}

var x = [];

funky(x);

alert(x);

What is x?

A. null

B.[]

C. undefined

D. throw

**Ans. B . []**

**Problem 2**

function swap(a,b) {

var temp = a;

a = b;

b = temp;

here we can avoid this part ,then result will be as it is.

}

var x =1, y=2;

swap(x,y);

alert(x);

What is x?

A.1

B.2

C.undefined

D. 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, add and mul, that take two numbers and return their sum and product.**

function add(x,y){

return x+y;

};

function mul(x,y){

return x\*y

}**;**

document.write(add(4,5)) ; // 9

document.write(mul(4,5)) ; // 20

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

function identity(n) {

return function() {

return n;

}

};

idf = identity(3);

document.write(idf());

**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 makes it callable with two invocations.**

function applyf(binary) {

return function(x) {

return function(y) {

return binary(x, y);

}

}

};

function add(x, y) {

return x + y + "<br>"

}

function mul(x, y) {

return x \* y;

}

addf = applyf(add);

addy = applyf(mul);

document.write("This is Addition of two indication: " + addf(3)(4)); //7

document.write("This is Multiplication of two indication: " + addy(3)(4)); //12

**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)

};

};

function add(x, y) {

return x + y + "<br>";

}

function mul(x, y) {

return x \* y;

}

add3 = curry(add, 3);

document.write("This will add 3 into an argument: " + add3(4)); //7

mul5 = curry(mul, 5);

document.write("This will multiply 5 with an argument: " + mul5(6)); // 30

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

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 args

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)**

function demethodize(func){

return function(x,y){

return func(x,y);

}

};

demethodize(Number.prototype.add)(5,6);

// 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(binary) {

return function(x) {

return binary(x, x);

};

};

function add(x, x) {

return x + x + "<br>";

};

function mul(x, x) {

return x \* x;

};

var double = twice(add);

document.write(double(11)); //22

var square = twice(mul);

document.write(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 twice(binary) {

return function(x) {

return binary(x, x);

}

}

function add(x, x) {

return x + x;

}

function mul(x, x) {

return x \* x;

}

var double = twice(add);

var square = twice(mul);

function compseu(f, g) {

return function(x) {

return g(f(x));

}

}

document.write(compseu(double, square)(3)); //36

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

function add(x, y) {

return x + y;

}

function mul(add, z) {

return add \* z;

}

function compseb(f, g) {

return function(x, y, z) {

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

}

}

document.write(compseb(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)

}

}

function add(x, y) {

return x + y;

}

var add\_once = once(add);

document.write(add\_once(3, 4)); //7

document.write(add\_once(3, 4)); // throw , because now add func 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 + "<br>";

},

dec: function() {

x -= 1;

return x-1;

}

}

}

counter = counterf(10);

document.write(counter.inc()); //11

document.write(counter.dec()); //9

**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);

document.write(temp.invoke(7));

document.write(temp.revoke());

document.write(temp.invoke(8));

// 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;

};

};

function add(x, y) {

return x + y + "<br>";

}

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

document.write(add\_ltd(3, 4)); //7

document.write(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++ + "<br>";

};

}

var index = from(0);

document.write(index()); //0

document.write(index()); //1

document.write(index()); //2

document.write(index()); //3

**Problem 20: 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 + "<br>";

}

return undefined;

};

}

function from(x) {

return function() {

return x++;

}

}

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

document.write(index()); //1

document.write(index()); //2

document.write(index()); //undefined

**Problem 21: 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 + "<br>";

}

return undefined;

};

}

var index = fromTo(0, 3);

document.write(index()); //1

document.write(index()); //2

document.write(index()); //3

document.write(index()); //undefined

**Problem 22: 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];

}

};

};

function fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start + "<br>";

}

return undefined;

};

};

var ele = element([

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

], fromTo(1, 3));

document.write(ele()); //a

document.write(ele()); //b

document.write(ele()); //undefined

**Problem 23: 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 fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start + "<br>";

}

return undefined;

};

};

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'

]);

document.write(ele());

document.write(ele());

document.write(ele());

document.write(ele());

**Problem 24: 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;

};

}

function fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start + "<br>";

}

return undefined;

};

}

var array = [],

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

document.write(col()); //1

document.write(col()); //2

document.write(col()); //undefined

**Problem 25: 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;

};

};

function fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start;

}

return undefined;

};

};

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

function third(value) {

return (value % 3) === 0;

});

document.write(fil());

document.write(fil());

document.write(fil());

**Problem 26: 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;

};

};

function fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start;

}

return undefined;

};

};

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

fromTo(0, 2));

document.write(con());

document.write(con());

document.write(con());

document.write(con());

document.write(con());

**Problem 27: Write a repeat function that takes a generator and calls it until it returns undefined.**

function repeat(gen) {

if (gen() !== undefined) {

return repeat(gen);

}

}

function fromTo(start, end) {

return function() {

if (start < end) {

start++;

return start;

}

return undefined;

};

};

var array = [];

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

document.write(log(array));

**Problem 28: 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;

};

document.write(map([2, 1, 0], inc));

**Problem 29: 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;

};

document.write(reduce([], add)); // undefined

document.write(reduce([2], add)); // 2

document.write(reduce([2, 1, 0], add)); // 3

**Problem 30: Make a function gensymf that makes a function that generates unique symbols.**

function from(x) {

return function() {

return x++ + "<br>";

};

};

function gensymf(sym) {

var number = from(1);

return function() {

return sym + number();

};

};

var geng = gensymf("G"),

genh = gensymf("H");

document.write(geng());

document.write(genh());

document.write(geng());

document.write(genh());

document.write(geng());

document.write(genh());

**Problem 31: 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;

};

};

};

function from(x) {

return function() {

return x++;

}

};

function gensymf(sym) {

var number = from(1);

return function() {

return sym + number();

};

};

var gensymf = gensymff(inc, 0),

geng = gensymf("G"),

genh = gensymf("H");

document.write(geng());

document.write(genh());

document.write(geng());

document.write(genh());

document.write(geng());

document.write(genh());

**Problem 32: 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;

}

};

};

var fibonaccik = fibonaccif(1, 4);

document.write(fibonaccik());

**// extra credit**

function fibonaccif(first,second){

return function(){

var next = first;

first = second;

second += next;

return next;

};

}

**Problem 33: 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: function() {

num += 1;

return num + "<br>";

},

down: function() {

num -= 1;

return num + "<br>";

}

}

};

var object = counter(10),

up = object.up,

down = object.down;

document.write(up());

document.write(down());

document.write(down());

document.write(up());

**Problem 34: 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"))

**Problem 35: 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")));

**Problem 36: 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 37: 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 38: Write a function exp that evaluates simple 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]

]

];

document.write(exp(nae)); // 5

**Problem 39: 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 + "<br>";

};

};

document.write(addg()); // undefined

document.write(addg(2)()); // 2

document.write(addg(2)(7)()); // 9

document.write(addg(3)(0)(4)()); // 7

document.write(addg(1)(2)(4)(8)()); // 15

**Problem 40: 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;

};

};

}

function mul(x, y) {

return x \* y;

}

document.write(liftg(mul)()); // undefined

document.write(liftg(mul)(3)()); // 3

document.write(liftg(mul)(3)(0)(4)()); // 0

document.write(liftg(mul)(1)(2)(4)(8)()); // 64

**Problem 41: 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);

};

document.write(arrayg()); // []

document.write(arrayg(3)()); // [3]

document.write(arrayg(3)(4)(5)()); // [3, 4, 5]

**Problem 42: 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];

},

};

}

document.write(myvector = vector());

document.write(myvector.append(7));

document.write(myvector.store(1, 8));

document.write(myvector.get(0)); // 7

document.write(myvector.get(1)); // 8

**Problem 43: 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();

document.write(my\_pubsub.subscribe(log));

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

// log("It works!")