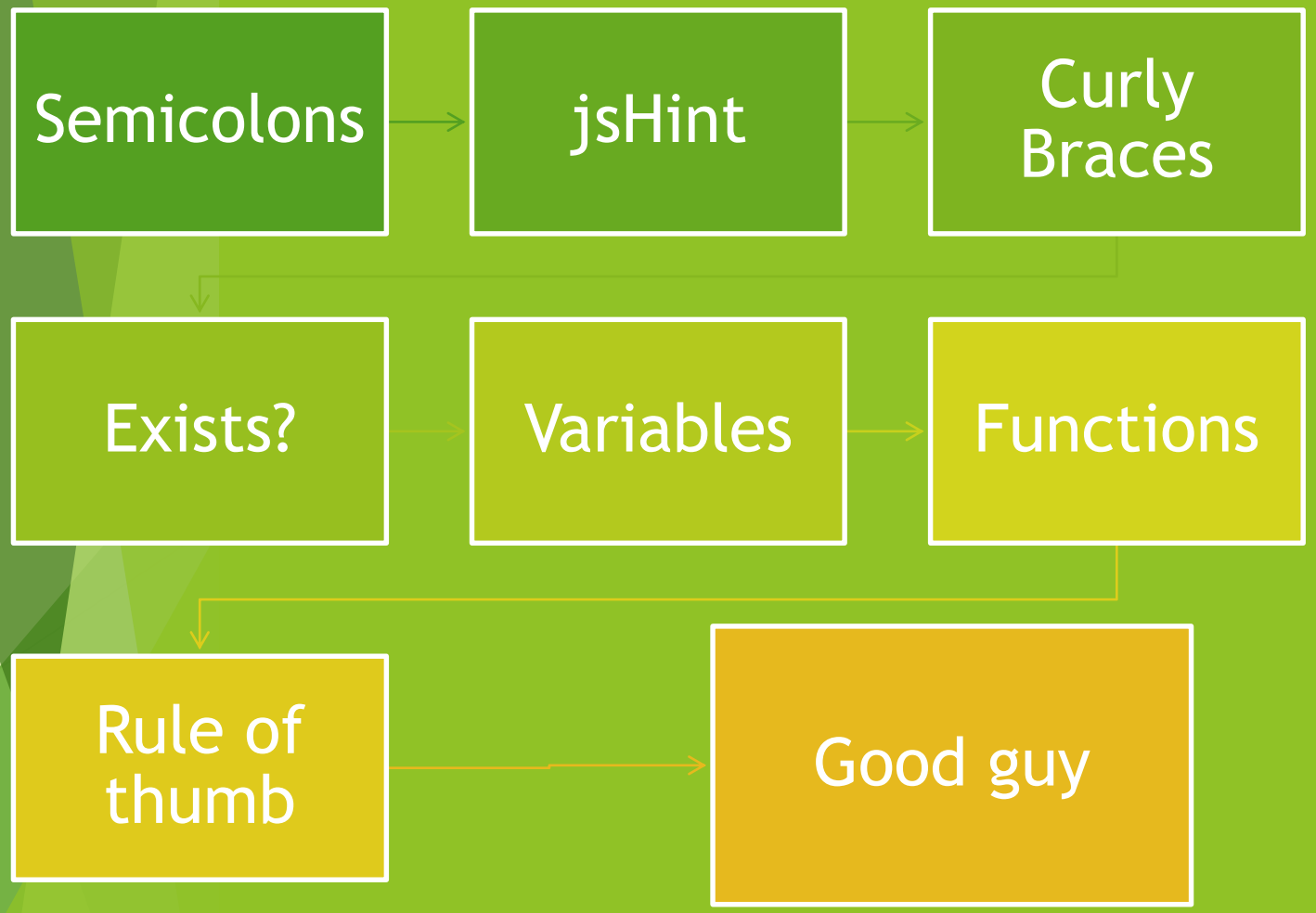

ACHIEVEMENT UNLOCKED
Reach 2 semester

Syntax

Syntax


Semicolons
jsHint
Curly Braces
Exists?
Variables
Functions
Rule of thumb
Good guy

Semicolons

“Semicolons are optional”

– *the people*

“Certain ECMAScript statements (variable statement, expression statement, do-while statement, continue statement, break statement, return statement, and throw statement) ***must be terminated*** with semicolons. ”

– *the facts*

“For convenience, however, such semicolons may be omitted from the source text in certain situations.”

– *the facts*


RULES
1.
2.
3.

“When, as the program is parsed from left to right, a token (called the offending token) is encountered that is not allowed by any production of the grammar, then a semicolon is automatically inserted before the offending token”

– *the facts*

```
var t = 1
var r = 4

if(true){console.log(t)}
```

“When, as the program is parsed from left to right, the end of the input stream of tokens is encountered, then a semicolon is automatically inserted at the end of the input stream”

– *the facts*

```
console.log(r)
```

```
var x =1;
var y =5;
var d = x + y
[1,2,3].foreach(e => console.log(e))
```

```
var x =1;
var y =5;
var d = x + y
(function(){
console.log('call');
})();
```

```
var x = [1,2,3]
var t = x
[1].toString();
console.log(t);
```

Continue, Break, return …

"When, [illegible] that is allowed by some production of the grammar, but the production is a restricted production and the token would be the first token of a restricted production and the restricted token is separated from the previous token by at least one LineTerminator, then a semicolon is automatically inserted before the restricted token"

– *the facts*

```
function semicolonTest()
{
    return
    {
        test: 1
    }
}

console.log(semicolonTest());
```

```
function example()
{
    var get = function()
    {
        console.log('get');
    }

    return
    {
        get: get
    }
}
```

```
function example(){
   var get = function(){
      console.log('get');
   };

   return{
      get: get
   };
}
```

jsHint

Exists?

```
var x;
if(x){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
var x=1;
if(x){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
var x=0;
if(x){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
if(x){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
var x;
if(typeof x !== 'undefined’){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
if(typeof x !== 'undefined’){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

```
var x;
if(typeof x !== undefined){
   console.log('X exists');
}
else{
   console.log('X does not exists');
}
```

Variables

“A var statement declares variables that are scoped to the running execution context's VariableEnvironment. Var variables are created when their containing Lexical Environment is instantiated and are initialized to undefined when created.”

– *the facts*

```
console.log(r);
```

```
console.log(r);
var r;
```

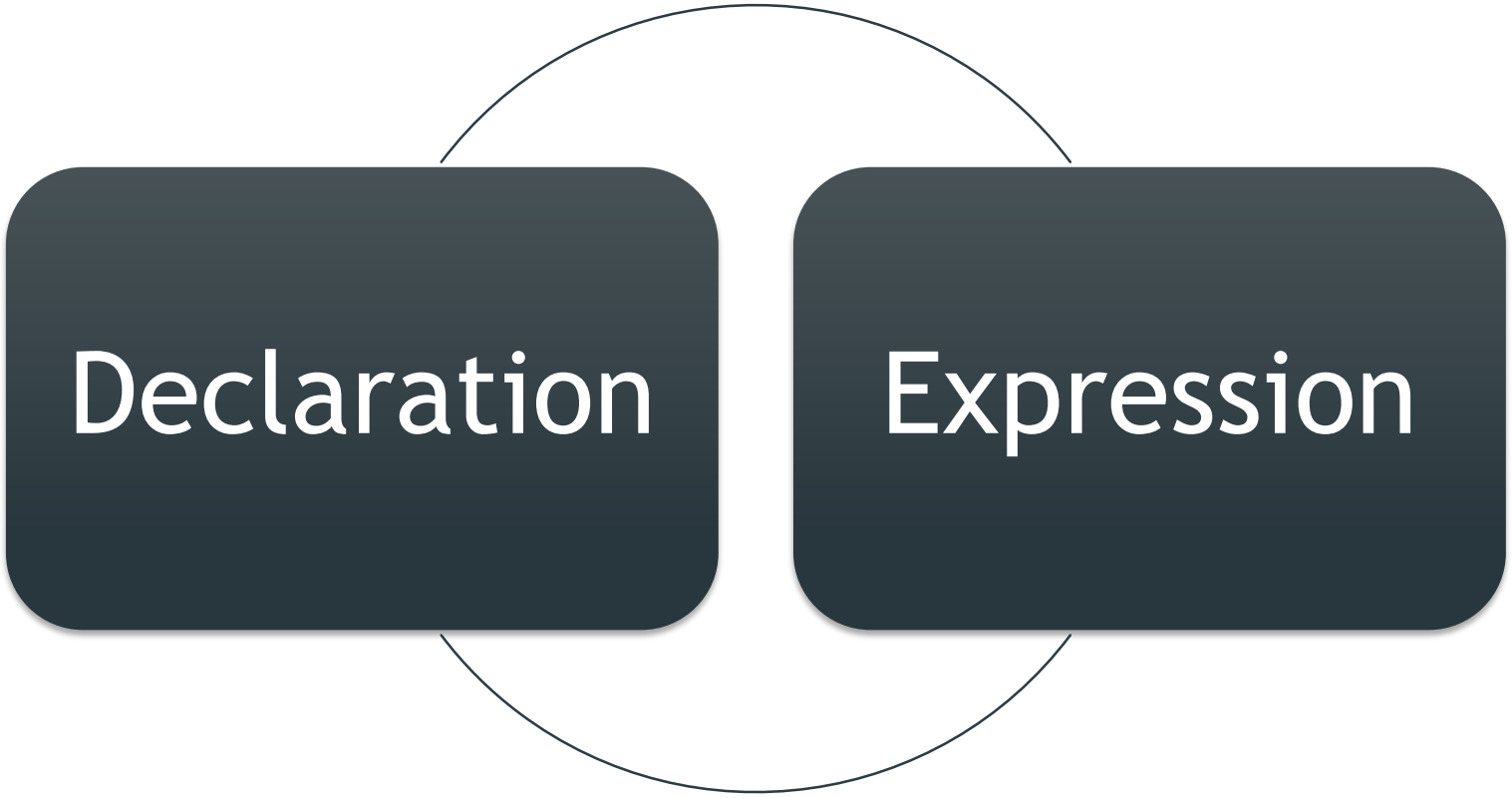
```
console.log(r);
var r=10;
```

```
var myVar = 10;

function myfun(){
   myVar = 11;
}

console.log(myVar);
```

```
var myVar = 10;

function myfun(){
   myVar = 11;
}

myfun();

console.log(myVar);
```

```
var myVar = 10;

function myfun(){
   myVar = 11;
   var myVar;
}

myfun();

console.log(myVar);
```

Functions



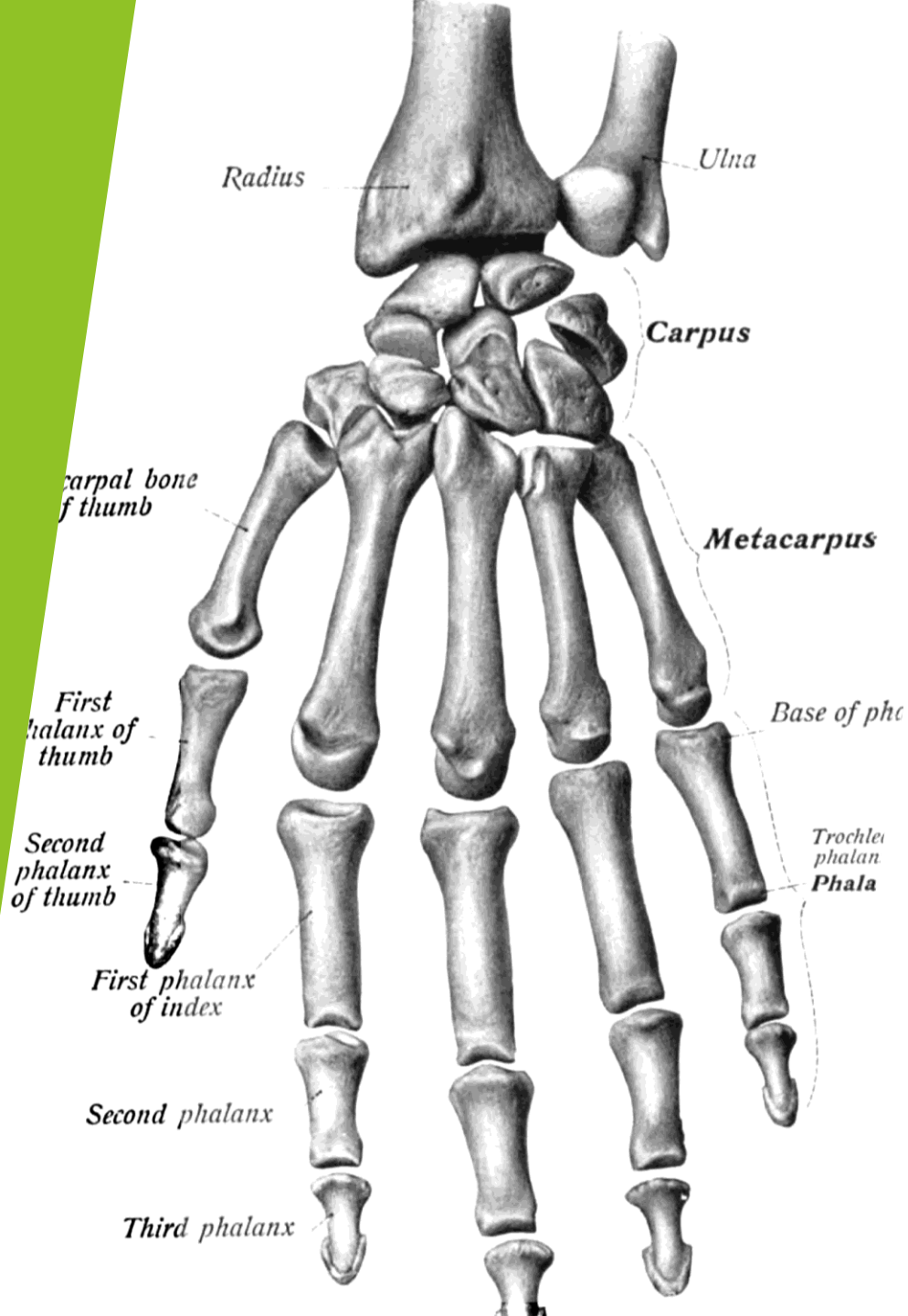
Functions

```
function declarationFunc(){
   console.log('declarationFunc');
}

declarationFunc();
```

```
declarationFunc();

function declarationFunc(){
   console.log('declarationFunc');
}
```

```
var expresionFunc = function(){
   console.log('expresionFunc');
};

expresionFunc();
```

```
expresionFunc();

var expresionFunc = function(){
   console.log('expresionFunc');
};
```



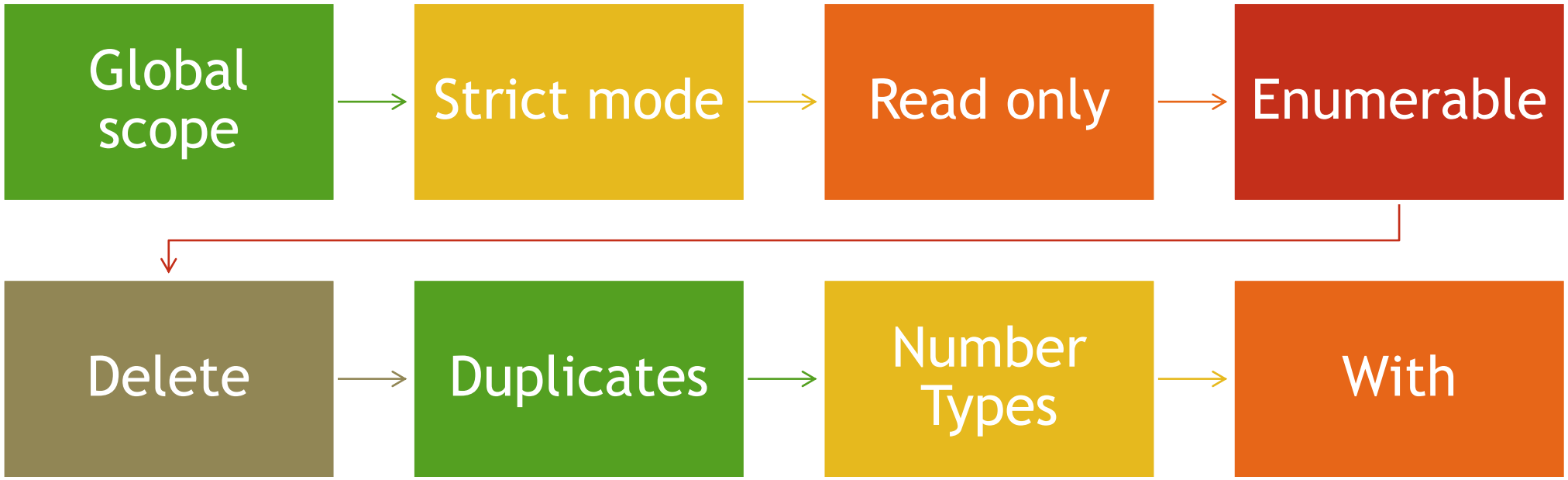
Rule of Thumb

- Variables
- Functions
- Code

Good guy

Behaviors

Behaviors


Global scope
Strict mode
Read only
Enumerable
Delete
Duplicates
Number Types
With

Global scope

```
function show(param){
   var innerParam = param;
   console.log(param);
}

show('test');
```

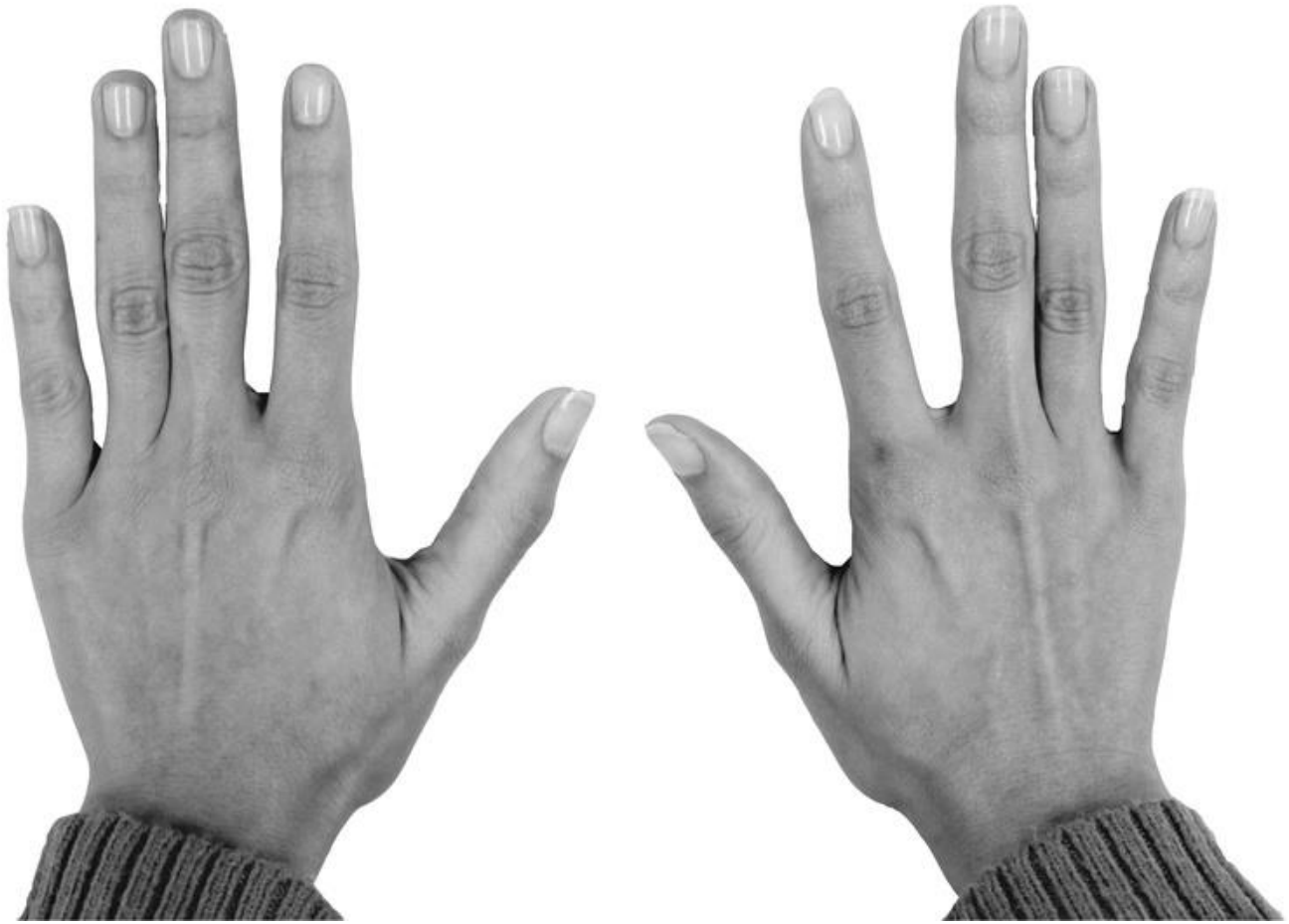
```
var val1 = 'show’ ;

function show(param){
   var innerParam = param;
   console.log(param);
   console.log(val1);
}

show('test');
```

```
function show(param){
   var innerParam = param;
   console.log(param);
}

console.log(innerParam);

show('test');
```

```
function show(param){
   var innerParam = param;
   console.log(param);
}

show('test’);

console.log(innerParam);
```

```
function show(param){
   innerParam = param;
   console.log(param);
}

console.log(innerParam);

show('test');
```

```
function show(param){
   innerParam = param;
   console.log(param);
}

show('test');
console.log(innerParam);
```



Strict mode

JavaScript please …. just stop helping !!!

```
'use strict’;

function show(param){
   innerParam = param;
   console.log(param);
}

show('test’);

console.log(innerParam);
```

```
function show(param){
   'use strict’;
   innerParam = param;
   console.log(param);
}

show('test’);

console.log(innerParam);
```

```
function show(param){
   'use strict’;
   var innerParam = param;
   console.log(param);
}

show('test’);

notCreatedVariable = 5;
```

```
'use strict’;

notCreatedVariable = 5;
```

```
'use strict’;

var obj = {};

obj.a = 'sdfs’;

console.log(obj);
```

Read only

```
var obj = {};

Object.defineProperty(obj, 'ro',{
   enumerable: true,
   configurable: true,
   writable: false,
   value: 'Original Value'
});

console.log(obj.ro);
```

```
var obj = {};

Object.defineProperty(obj, 'ro',{
   enumerable: true,
   configurable: true,
   writable: false,
   value: 'Original Value'
});

obj.ro = 'Altered Value’;

console.log(obj.ro);
```

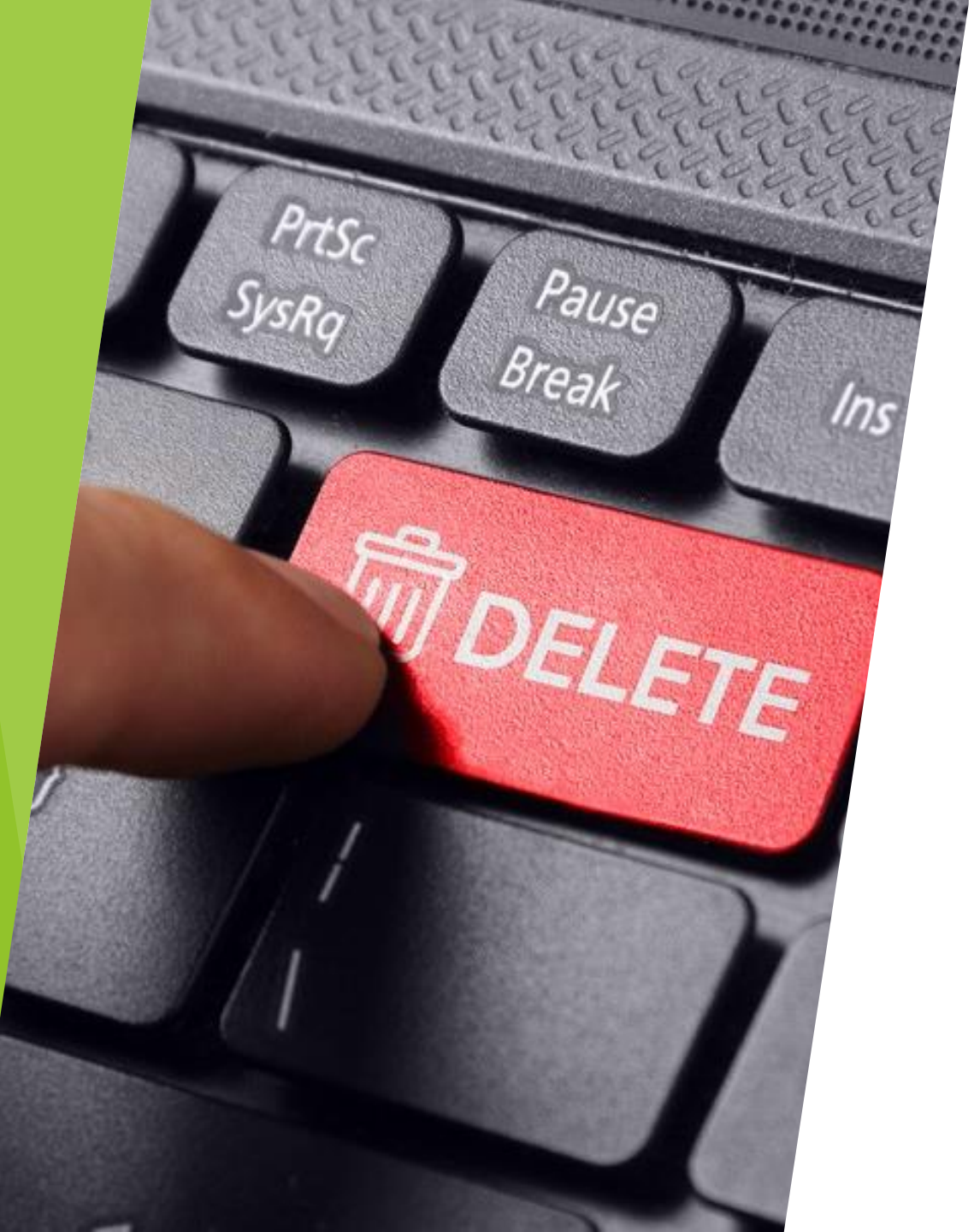
```
'use strict’;

var obj = {};

Object.defineProperty(obj, 'ro',{
enumerable: false,
configurable: false,
writable: false,
value: 'Original Value'
});

obj.ro = 'Altered Value’;

console.log(obj.ro);
```

Enumerable

```
var obj = {
    c : 'C Value’
};

obj.a = 'A Value’;

Object.defineProperty(obj, 'b',{
    enumerable: true,
    configurable: true,
    writable: true,
    value: 'B Value'
});

console.log(obj);
for (var key in obj) {
    console.log(key);
}
```

```
var obj = {
    c : 'C Value’
};

obj.a = 'A Value’;

Object.defineProperty(obj, 'b',{
    enumerable: false,
    configurable: true,
    writable: true,
    value: 'B Value'
});

console.log(obj);
for (var key in obj) {
    console.log(key);
}

console.log(obj.b);
```

```
var obj = {
    c : 'C Value’
};

obj.a = 'A Value’;

    Object.defineProperty(obj, 'a',{
    enumerable: false
});

Object.defineProperty(obj, 'b',{
    enumerable: false,
    configurable: true,
    writable: true,
    value: 'B Value'
});

console.log(obj);
for (var key in obj) {
    console.log(key);
}

console.log(obj.a);
console.log(obj.b);
```



Delete

```
var obj = {
   a:'A’,
   b:'B’
};

console.log(obj);
```

```
var obj = {
   a:'A’,
   b:'B’
};

delete obj.b;

console.log(obj);
```

```
var x = 6;

delete x;

console.log(x);
```

```
var obj = {
   a:'A’,
   b:'B’
};

delete obj;

console.log(obj);
```

```
'use strict’;

var x = 6;

delete x;

console.log(x);
```

```
'use strict’;

var obj = {
   a:'A’,
   b:'B’
};

delete obj;

console.log(obj);
```

Duplicates

```
function test(x,y,x){
   console.log(x);
}

test('a','b','c');
```

```
'use strict’;

function test(x,y,x){
   console.log(x);
}

test('a','b','c');
```

Number Types

```
var x = 120,
y = 012;

console.log(x+y);
```

```
var x = 120,
y = 0x12;

console.log(x+y);
```

```
'use strict’;

var x = 120,
y = 012;

console.log(x+y);
```

```
'use strict’;

var x = 120,
y = 0x12;

console.log(x+y);
```

```
'use strict’;

var x = 120,
y = parseInt(12,8);

console.log(x+y);
```

with

```
var obj = {
   a:{
      b:'B'
   }
};

console.log(obj.a.b);
```

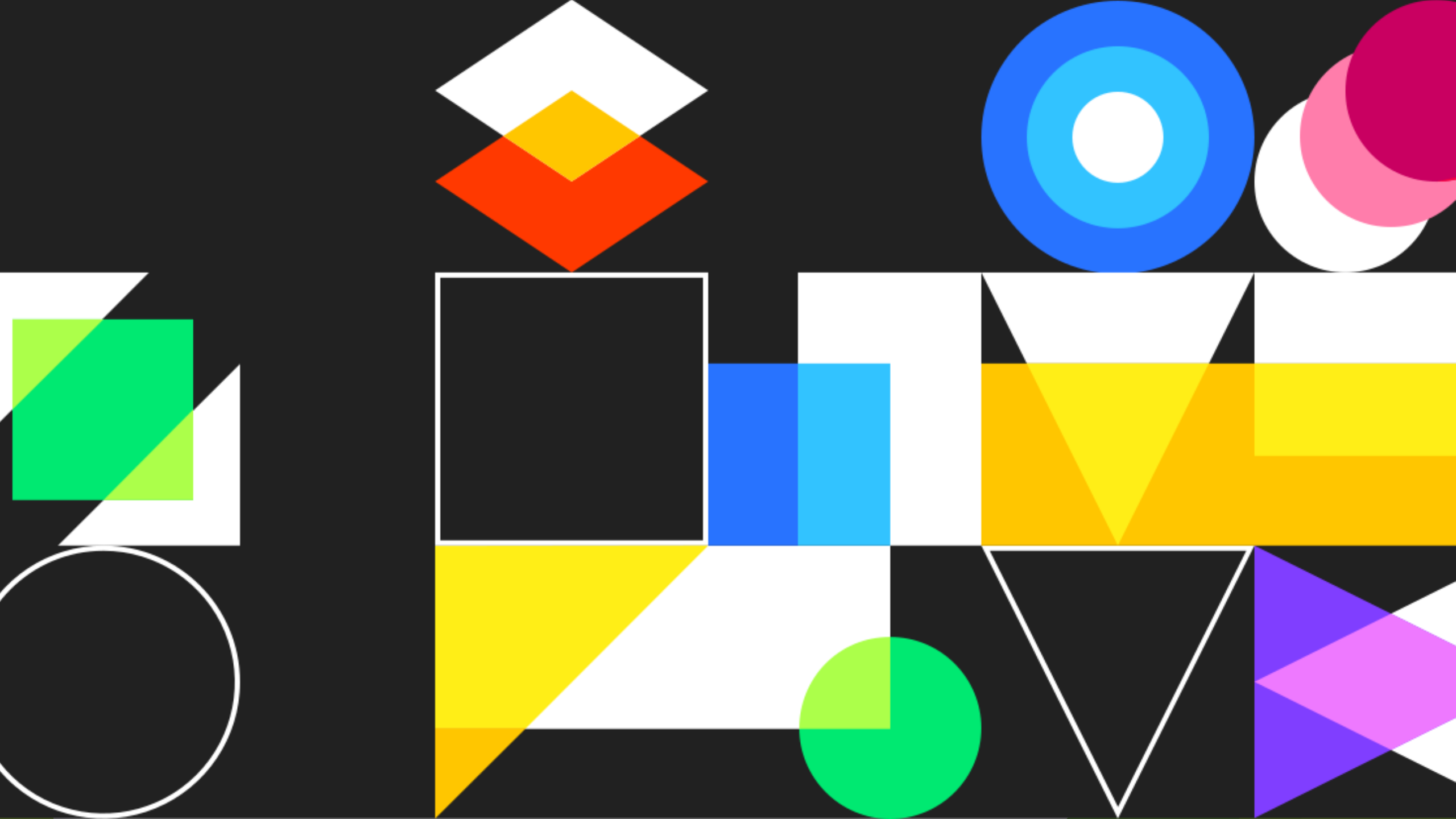
```
var obj = {
   a:{
      b:'B'
   }
};

with(obj.a){
   console.log(b);
}
```

```
var obj = {
   a:{
      b:'B'
   }
};

var b = 'C’;

with(obj.a){
   console.log(b);
}
```

```
'use strict’;

var obj = {
   a:{
      b:'B'
   }
};

var b = 'C’;

with(obj.a){
   console.log(b);
}
```

```
'use strict’;

var obj = {
   a:{
      b:'B'
   }
};
var b = 'C’;

(function(tempVar){
   console.log(tempVar);
}(obj.a.b));
```

Design Patterns



Design Patterns

- Modules x3
- Singleton x2
- Factory x2
- Decorator x4

Module / Closure

```
const myModule = (function() {
    const privateVariable = "Hello World";

    function privateMethod() {
        console.log(privateVariable);
    }

    return {
        publicMethod: function() {
            privateMethod();
        }
    };
})();

console.log(myModule);

myModule.publicMethod();
```

Revealing Module ½ (definition)

```
const myRevealingModule = (function() {
    let privateVar = "Peter";
    const publicVar = "Hello World";

    function privateFunction() {
        console.log("Name: " + privateVar);
    }

    function publicSetName(name) {
        privateVar = name;
    }

    function publicGetName() {
        privateFunction();
    }

    return {
        setName: publicSetName,
        greeting: publicVar,
        getName: publicGetName
    };
})();
```

Revealing Module 2/2 (usage)

```
console.log(myRevealingModule);

myRevealingModule.setName("Mark");
myRevealingModule.getName();
```

ES6 Module 1/3 (definition)

```
const greeting = "Hello World";
let callCounter = 0;

function sayHelloFirstTime() {
    if (callCounter++ === 0) {
        console.log(greeting);
    }
}

function privateLog() {
    console.log("Private Function");
}

function multiply(num1, num2) {
    sayHelloFirstTime();
    console.log("Multiply:", num1, num2);
    return num1 * num2;
}
```

ES6 Module 2/3 (definition)

```
function subtract(num1, num2) {
    sayHelloFirstTime();
    console.log("Subtract:", num1, num2);
    return num1 - num2;
}

module.exports = {
    sum: sum,
    subtract: subtract,
    multiply: multiply,
    divide: divide
};
```

ES6 Module 3/3 (usage)

```
let mathUtils = require("./utils/utils");

console.log(mathUtils);

console.log(mathUtils.sum(3, 7));
console.log(mathUtils.subtract(3, 7));
console.log(mathUtils.multiply(3, 7));
console.log(mathUtils.divide(3, 7));
```

Singleton

```
function User() {
    this.name = "Peter";
    this.age = 25;
}

const user1 = new User();
const user2 = new User();
console.log(user1 === user2);
```

Singleton

```
let instance = null;

function User() {
    if (instance) {
        return instance;
    }

    instance = this;
    this.name = "Peter";
    this.age = 25;

    return instance;
}

const user1 = new User();
const user2 = new User();
console.log(user1 === user2);
```

Singleton

```
const singleton = (function() {
    let instance;

    function init() {
        return {
            name: "Peter",
            age: 24
        };
    }
    return {
        getInstance: function() {
            if (!instance) {
                instance = init();
            }

            return instance;
        }
    };
})();

const instanceA = singleton.getInstance();
const instanceB = singleton.getInstance();
console.log(instanceA === instanceB);
```

Factory Basic ½ (definition)

```
class Car {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "brand new";
        this.color = options.color || "white";
    }
}
class Truck {
    constructor(options) {
        this.doors = options.doors || 4;
        this.state = options.state || "used";
        this.color = options.color || "black";
    }
}
class VehicleFactory {
    createVehicle(options) {
        if (options.vehicleType === "car") {
            return new Car(options);
        } else if (options.vehicleType === "truck") {
            return new Truck(options);
        }
    }
}
```

Factory Basic 2/2 (usage)

```
const factory = new VehicleFactory();

const car = factory.createVehicle({
    vehicleType: "car",
    doors: 4,
    color: "silver",
    state: "Brand New"
});

const truck = factory.createVehicle({
    vehicleType: "truck",
    doors: 2,
    color: "white",
    state: "used"
});

console.log(car);

console.log(truck);
```

Factory Advanced 1/3 (objects definition)

```
class Vehicle {
    constructor(vehicleType) {
        this.vehicleType = vehicleType;
    }
}

class Car extends Vehicle {
    constructor(options) {
        super(options.vehicleType);
        this.doors = options.doors || 4;
        this.state = options.state || "brand new";
        this.color = options.color || "white";
    }
}

class Truck extends Vehicle {
    constructor(options) {
        super(options.vehicleType);
        this.doors = options.doors || 4;
        this.state = options.state || "used";
        this.color = options.color || "black";
    }
}
```

Factory Advanced 2/3 (factory definition)

```
class VehicleFactory {
   createVehicle(options) {
      if (options.vehicleType === "car") {
         return new Car(options);
      } else if (options.vehicleType === "truck")
      {
         return new Truck(options);
      }
   }
}
```

Factory Advanced 3/3 (usage)

```
const factory = new VehicleFactory();

const car = factory.createVehicle({
    vehicleType: "car",
    doors: 4,
    color: "silver",
    state: "Brand New"
});

const truck = factory.createVehicle({
    vehicleType: "truck",
    doors: 2,
    color: "white",
    state: "used"
});

console.log(car);

console.log(truck);
```

Decorator 1/3

```
class Car {
   constructor() {
      this.cost = function() {
         return 20000;
      };

      this.desc = "basic";
   }
}
```

Decorator 2/3

```
function carWithAC(car) {
    car.hasAC = true;
    const prevCost = car.cost();
    car.cost = function() {
        return prevCost + 500;
    };

    car.desc += " AC";
}

function carWithAutoTransmission(car) {
    car.hasAutoTransmission = true;
    const prevCost = car.cost();
    car.cost = function() {
        return prevCost + 2000;
    };

    car.desc += " AutoT";
}
```

Decorator 3/3

```
const car = new Car();
console.log(car.cost());
carWithAC(car);
carWithAutoTransmission(car);

console.log(car.cost());
console.log(car.desc);
console.log(car);
```

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
        };

        this.desc = "basic";
    }
}

const car = new Car();
car.autoPark = () => console.log("Auto parking...");

console.log(car);
car.autoPark();
```

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
        };

        this.desc = "basic";
    }
}

function addParking(car) {
    car.autoPark = () => console.log("Auto parking...");
}

const car = new Car();

addParking(car);
console.log(car);
car.autoPark();
```

Decorator

```
class Car {
    constructor() {
        this.cost = function() {
            return 20000;
    };

        this.desc = "basic";
    }
}

function addParking() {
    Car.prototype.autoPark = () => console.log("Auto parking...");
}

const car = new Car();
addParking();

console.log(car);
car.autoPark();
```