Inheritance ES5

1. with Object.create and constructor as prototype chain <https://jsfiddle.net/zLbn3gvo/2/>

function Parent() { this.pname = 'abc'; }

function Child() {

Parent.call(this);

this.cname = 'xyz'; }

Child.prototype = Object.create(Parent.prototype);

Child.prototype.constructor = Child;

var a = new Child;

console.log(a.pname); // abc

console.log(a.cname); // xyz

1. with prototype and new by object/instance and function constructor <https://jsfiddle.net/zLbn3gvo/3/>

function Parent() { this.pname = 'abc'; }

function Child() { this.cname = 'xyz'; }

Child.prototype = new Parent();

var a = new Child;

console.log(a.pname); // abc

console.log(a.cname); // xyz

1. with call(this, arguments) <https://jsfiddle.net/zLbn3gvo/4/>

function Parent() { this.pname = 'abc'; }

function Child() {

Parent.call(this);

this.cname = 'xyz';

}

var a = new Child;

console.log(a.pname); // abc

console.log(a.cname); // xyz

Inheritance ES6

1. ES6 with increment <http://jsfiddle.net/wveh0dru/1/>

class Parent {

constructor() { this.pnum = 10; }

sayHello() { console.log('Hello'); }

}

class Child extends Parent {

constructor() {

super();

this.cnum = 20;

}

mathNum () {

this.pnum++;

this.cnum--;

this.sayHello();

console.log(this.pnum);

console.log(this.cnum);

}

}

let a = new Child();

console.log(a.pnum);

console.log(a.cnum);

console.log(a.sayHello());

console.log(a.mathNum());

1. ES6 with passing arguments in constructor <http://jsfiddle.net/wveh0dru/2/>

class Parent {

constructor(fname, lname) {

this.fname = fname || 'abc';

this.lname = lname || 'xyz';

}

sayHello() { alert('Hello'); //console.log('Hello'); }

}

class Child extends Parent {

constructor(fname, lname) {

super(fname, lname);

this.cname = 'mno';

}

}

let a = new Child();

console.log(a.fname);

console.log(a.lname);

console.log(a.cname);

console.log(a.sayHello());

Closure

closures are dealing with two realities of JavaScript:

a. scope is at the function level, not the block level and,

b. much of in JavaScript is asynchronous/event driven.

A closure is one way of supporting [first-class functions](https://en.wikipedia.org/wiki/First-class_function); it is an expression that can reference variables within its scope (when it was first declared),

-> be assigned to a variable,

-> be passed as an argument to a function, or

-> be returned as a function result.

1. example <https://jsfiddle.net/zLbn3gvo/>

// define a function that returns a function as variable (say1 /say2).

function test1(name) {

var text = "hello " + name, i = 0; // local var

var result = function() { // create result

i++;

console.log(text + " " + i); //alert(text + " " + i);

};

return result;

}

// get two function results

say1 = test1('Dick');

say2 = test1('Jane');

say1(); // hello Dick 1

say2(); // hello Jane 1

say2(); // hello Jane 2

say2(); // hello Jane 3

say1(); // hello Dick 2

-->The alert/console-log function has access to the code within the test1() method scope. So each function is a pointer to an instance.

2 example <https://jsfiddle.net/zLbn3gvo/1/>

function sayHello(name) {

var text = 'Hello ' + name;

var say = function() { console.log(text); }

say();

}

sayHello('Joe'); // Hello Joe

3 examples <https://jsfiddle.net/zLbn3gvo/6/>

4 anonymous functions after it declared a variable as [variable hoisting](https://stackoverflow.com/a/3725763/1269037)  <https://jsfiddle.net/zLbn3gvo/8/>

function sayHello() {

var say = function() { console.log(hello); }

var hello = 'Hello World!'; // Local variable that ends up within closure

return say;

}

sayHello()(); // "Hello World!"

--> The say variable is also inside the closure and could be accessed by any other function that might be declared within sayHello(), or it could be accessed recursively within the inside function.

Links: <https://stackoverflow.com/questions/111102/how-do-javascript-closures-work>

5 Closure Angular/ Emulating private methods <https://jsfiddle.net/zLbn3gvo/9/>

var counter = (function () {

var num = 0;

function ab(val) { num += val; }

return {

increment: function() { ab(1); },

decrement: function() { ab(-1); },

value: function() { return num; }

};

}) ();

console.log(counter.value()); //logs 0

counter.increment();

counter.increment();

console.log(counter.value()); //logs 2

counter.decrement();

console.log(counter.value()); //logs 1

write a function to receive URL as an input and print out the domain name, path, query-string, protocol. <https://jsfiddle.net/wgkosabz/6/>

function parseURL(url) {

var a = document.createElement('a');

a.href = url;

return {

host: a.hostname,

path: a.pathname,

query: a.search,

protocol: a.protocol

};

}

function showResult(data) {

var skeleton = "";

skeleton += "<ol>";

skeleton += "<li>Domain name/Host name: " + data.host + "</li>";

skeleton += "<li>Path/Path name: " + data.path + "</li>";

skeleton += "<li>Query-string/Search: " + data.query + "</li>";

skeleton += "<li>Protocol: " + data.protocol + "</li>";

skeleton += "</ol>";

document.getElementById('result').innerHTML = skeleton;

}

function clickParseURL() {

showResult(parseURL(document.getElementById('urlStr').value));

}

//some more

return {

source: url,

protocol: a.protocol.replace(':', ''),

host: a.hostname,

port: a.port,

query: a.search,

params: (function () {

var ret = {},

seg = a.search.replace(/^\?/, '').split('&'),

len = seg.length,

i = 0,

s;

for (; i < len; i++) {

if (!seg[i]) {

continue;

}

s = seg[i].split('=');

ret[s[0]] = s[1];

}

return ret;

})(),

file: (a.pathname.match(/\/([^\/?#]+)$/i) || [, ''])[1],

hash: a.hash.replace('#', ''),

path: a.pathname.replace(/^([^\/])/, '/$1'),

relative: (a.href.match(/tps?:\/\/[^\/]+(.+)/) || [, ''])[1],

segments: a.pathname.replace(/^\//, '').split('/')

};