JAVASCRIPT FUNCTIONS

FUNCTION

Callable first class citizen (Can be passed and return as object)

No overloading

Definitions

- var add = new Function('a', 'b', 'return a + b');
- var add = function (a, b) { return a + b; };
- function add(a, b) { return a + b;}

Blessed with

- this
- arguments

JAVASCRIPT - LENGTH

Specifies the number of arguments expected by the function.

Syntax

- functionName.length

E.g.,

- **console.log((function () {}).length); // 0**
- **console.log((function (a) {}).length); // 1**
- **console.log((function (a, b) {}).length); // 2**

INVOCATION PATTERN I

Function invocation (Direct Invocation)

- add(1, 2)
- isPalindrome('madam')

this bound to global object !!!

INVOCATION PATTERN II

Method Invocation

Method => a function stored as property of object

this bound to method holder object

```
var obj = {
    value  :  0, //zero
    increment  :  function (inc) {
          this.value  +=  typeof  inc === 'number' ? inc : 1;
    }
}

obj.increment() ; // 1
obj.increment(2);  //  3
```

INVOCATION PATTERN III

Constructor Invocation (OO style)

```
var Employee = function (firstName, title) {
    this.firstName = firstName;
    this.title = title;
};

Employee.protoype.getName = function () { return this.name;};
Employee.protoype.getTitle = function () { return this.title;};

var  employee = new Employee('Tom', 'Software Engineer')
employee.getName();  // 'Tom'
employee.getTitle();     // 'Software Engineer'
```

INVOCATION PATTERN IV

Apply Invocation (Reflective Invocation)

```
var argsArray = [2, 3];
var sum = add.apply( null, argsArray);    // 3
var sum = add.call( null, 2, 3);          // 3

var firstName = Employee.getName.apply(empObject);
var firstName = Employee.getName.call(empObject);
```

JAVASCRIPT - CALL

Calls a function with a given this value and arguments provided individually.

Syntax

- fun.call(thisArg[, arg1[, arg2[, ...]]])

JAVASCRIPT - CALL

```
function diplayInfo(year, month, day){
  return "Name:" + this.name + ";birthday:" + year +
      "." + month + "." + day;
}

var p = { name: "Jason" };

diplayInfo.call(p, 1985, 11, 5);
```

JAVASCRIPT - APPLY

Calls a function with a given this value and arguments provided as an array

Syntax

- fun.apply(thisArg[, argsArray])

JAVASCRIPT - APPLY

```
function diplayInfo(year, month, day){
  return "Name:" + this.name + ";birthday:" + year +
  "." + month + "." + day;
}

var p = { name: "Jason" };

console.log(diplayInfo.apply(p, [1985, 11, 5]));
```

JAVASCRIPT - CALLEE

Specifies the currently executing function

callee is a property of the arguments object.

Syntax

- [function.]arguments.callee

JAVASCRIPT - CALLEE

```
function factorial(n){
  if (n <= 0)
     return 1;
  else
     return n * arguments.callee(n - 1);
}

factorial(4);
```

JAVASCRIPT - BIND

Creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

Syntax

fun.bind(*thisArg*[, *arg1*[, *arg2*[, ...]]])

JAVASCRIPT - BIND

```
var x = 9;
var module = {
  x: 81,
  getX: function() { return this.x; }
};

module.getX();   //Answer: ?

var getX = module.getX;
getX();    //Answer: ?

var boundGetX = getX.bind(module);
boundGetX();    //Answer: ?
module.x = 100;
boundGetX();  //Answer: ?
```

JAVASCRIPT - BIND

```
var checkNumericRange = function (value) {
    return value >= this.min && value <= this.max;
}

var range = { min: 10, max: 20 };

var boundCheckNumericRange = checkNumericRange.bind
(range);

var result = boundCheckNumericRange (12);
```

Result = ??

JAVASCRIPT - BIND

```
var displayArgs = function (val1, val2, val3, val4) {
    console.log(val1 + " " + val2 + " " + val3 + " " + val4);
}
var emptyObject = {};

var displayArgs2 = displayArgs.bind(emptyObject, 12, "a");

displayArgs2("b", "c");    //Answer: ?
```

JAVASCRIPT - BIND

```
Function.prototype.bind = function (objToBind) {
    var self = this;
  return function () {
    var argArr = Array.prototype.slice.call(arguments);
      return self.apply(objToBind || null, argArr);
  };
}
```

FUNCTION AS A CLASS

```
var someClass = function (property) {
    this.publicProperty = property;
    var privateVariable = "value";

    this.publicMethod = function () {
        //code for method definition
    };
    var privateMethod = function () {
        //code for method definition
    };
    // return this;
}
```

FUNCTION AS A MODULE

```
var counterModule = ( function( )  {
      var privateCount = 0;
      function changeBy(val) {
            return privateCount += val;
      }
      return {
            increment : changeBy.bind(null, 1),
            decrement : changeBy.bind(null, -1),
            currentValue : function() {return privateCount;}
      };
} ) ( );
```