



Universidad La Salle

Ingeniería de Software

Fundamentos de Lenguajes de Programación

Karlo Pacha Curimayhua

Fecha: Octubre 25, 2022

Práctica 13

Ejercicios

1. Investigue el concepto de first class en Javascript y muestre una pequeña deficiencia seguida de ejemplos. (2 puntos)
 - (a) Son funciones que son tratadas como cualquier otra variable; estas pueden ser pasadas como argumento a otras funciones, pueden ser retornadas por otra función y pueden ser asignadas a una variable.

```
1  const Arithmetics = {
2    add: (a, b) => {
3      return `${a} + ${b} = ${a + b}`;
4    },
5    subtract: (a, b) => {
6      return `${a} - ${b} = ${a - b}`;
7    },
8    multiply: (a, b) => {
9      return `${a} * ${b} = ${a * b}`;
10   },
11   division: (a, b) => {
12     if (b !== 0) return `${a} / ${b} = ${a / b}`;
13     return 'Cannot Divide by Zero!!!';
14   }
15 }
16
17 function sayHello() {
18   return "Hello, ";
19 }
```

```

20 function greeting(helloMessage, name) {
21   console.log(helloMessage() + name);
22 }
23 // Pass 'sayHello' as an argument to 'greeting' function
24 greeting(sayHello, "JavaScript!");
25 // Hello, JavaScript!

```

2. Describa la diferencia entre Currying and Partial Application. Incluya ejemplos. (2 puntos)

- (a) Currying: Una función que toma una función con múltiples parámetros como entrada y devuelve una función con exactamente un parámetro.
- (b) Partial Application: El proceso de aplicar una función a algunos de sus argumentos. La función aplicada parcialmente se devuelve para su uso posterior.

```

1 function partial(firstArgument, secondArgument) {
2   return function(thirdArgument, fourthArgument, fifthArgument) {
3     return firstArgument + secondArgument + thirdArgument +
4       fourthArgument + fifthArgument;
5   }
6 }
7 function curry(firstArgument) {
8   return function(secondArgument) {
9     return function(thirdArgument) {
10      return function(fourthArgument) {
11        return firstArgument + secondArgument + thirdArgument +
12          fourthArgument;
13      }
14    }
15  }
16 }

```

3. Implemente una función que calcule el volumen de un cilindro. Incluya la versión normal y una aplicando Currying. (2 puntos)

```

1 // Implemente una función que calcule el volumen de un cilindro.
  Incluya la versión normal y una
2 // aplicando Currying.
3
4 var NormalCylinderVolume = (r, h) => Math.PI * Math.pow(r,2) * h;
5
6 console.log("Normal: ", NormalCylinderVolume(2, 3));
7
8 var CurryingCylinderVolume =(r) => {

```

```

9     return (h) => Math.PI * Math.pow(r,2) * h;
10 }
11 console.log("Currying: ",CurryingCylinderVolume(2)(3));

```

```

Normal:  37.69911184307752
Currying:  37.69911184307752

```

4. Cree una función joinWords que una varios parametros de tipo string. (3 puntos)

```

1 // Cree una función joinWords que una varios parametros de tipo
  string.
2
3 function joinWords(string1) {
4     return (string2) => !string2 ? string1 : joinWords(`${string1}
    ${string2}`);
5 }
6
7 result = joinWords('Hello')();
8 console.log(result); // Hello
9
10 result = joinWords('There')('is')('no')('spoon.')();
11 console.log(result); // There is no spoon.

```

```

Hello
There is no spoon.

```

5. Implemente una función delayInvoc que en cada invocación incremente la variable total con el valor enviado como parametro. (3 puntos)

```

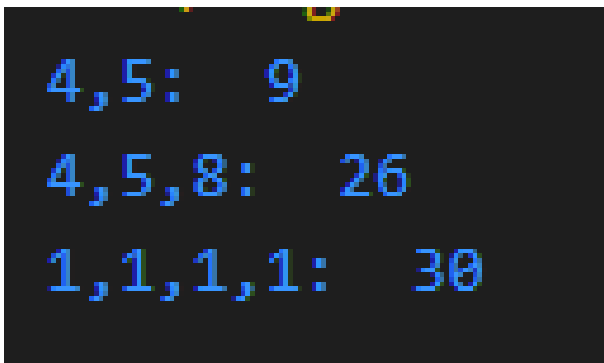
1 // Implemente una función delayInvoc que en cada invocación
  incremente la variable total con el
2 // valor enviado como parametro.
3
4 var total = 0;
5
6 var delayInvoc = function (a) {
7     total += a;

```

```

8     return function (b) {
9         if(b) return delayInvoc(b);
10    };
11 };
12
13
14 delayInvoc(4)(5);
15 console.log("4,5: ",total); //9
16
17 delayInvoc(4)(5)(8);
18 console.log("4,5,8: ",total); // 26
19
20 delayInvoc(1)(1)(1)(1);
21 console.log("1,1,1,1: ",total); // 30

```



```

4,5: 9
4,5,8: 26
1,1,1,1: 30

```

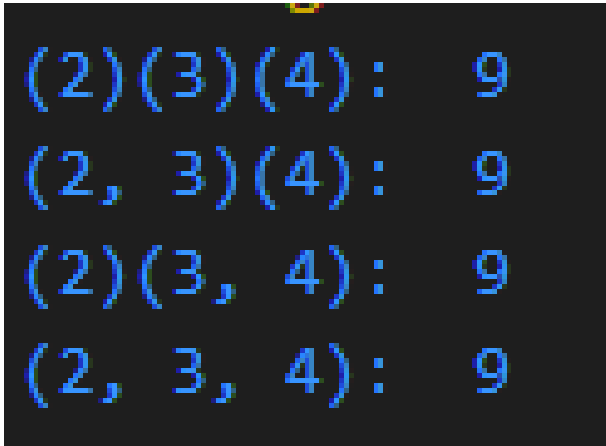
6. Implemente una función curry que tome como argumento cualquier función f y retorne la versión curried de f. (4 puntos)

```

1 // Implemente una función curry que tome como argumento
  cualquier función f y retorne la versión
2 // curried de f.
3
4 function abc(a, b, c) {
5     return a+b+c;
6 }
7
8 function curry(f) {
9     return function curry2(...args) {
10         if (args.length >= f.length) return f.apply(this, args);
11         else return function curry3(...args2) {
12             return curry2.apply(this, args.concat(args2));
13         }
14     };
15 }

```

```
16
17 var curriedAbc = curry(abc);
18
19 console.log("(2)(3)(4): ", curriedAbc(2)(3)(4)); // 9
20 console.log("(2, 3)(4): ", curriedAbc(2, 3)(4)); // 9
21 console.log("(2)(3, 4): ", curriedAbc(2)(3, 4)); // 9
22 console.log("(2, 3, 4): ", curriedAbc(2, 3, 4)) ; // 9
```



```
(2)(3)(4): 9
(2, 3)(4): 9
(2)(3, 4): 9
(2, 3, 4): 9
```

GitHub: <https://github.com/KEPCU/FundamentalsOfProgrammingLanguages/tree/master/js/practice13>