Hoja de ejercicios Tema 3

Programación en Javascript

▷ **1.** Escribe una función `producto` que reciba dos parámetros (llamados `x` e `y`) y devuelva su producto, teniendo en cuenta que tanto la `x` como la `y` pueden ser números o vectores (representados como arrays). La función se comportará del siguiente modo:

- Si `x` e `y` son números, se calculará su producto.
- Si `x` es un número e `y` es un vector (o viceversa), se calculará el vector que resulta de multiplicar todas los componentes de `y` por `x`.
- Si `x` e `y` son vectores de la misma longitud, se calculará el producto escalar de ambos.
- En cualquier otro caso, se lanzará una excepción.

▷ **2.** Implementa una función `sequence` que reciba un array de funciones `[f_1, ..., f_n]` y un elemento inicial `x`. La función debe aplicar `f_1` a `x`, pasar el resultado a `f_2`. A su vez, se pasará el resultado de `f_2` a `f_3` y así sucesivamente.

(a). En primer lugar, implementa `sequence` suponiendo que ninguna de las funciones del array recibido devuelve el valor `undefined`.

(b). Modifica la función anterior para que, en el caso en que una función del array devuelva el valor `undefined`, la función `sequence` devuelva directamente `undefined` sin seguir ejecutando las funciones restantes.

(c). Modifica la función para que reciba un tercer parámetro opcional (`right`), cuyo valor por defecto será `false`. Si el parámetro `right` tiene el valor `true`, el recorrido del elemento por las funciones será en orden inverso: desde la última función del array hasta la primera.

▷ **3.** Supongamos el siguiente fragmento de código *Javascript*:

```
/*
 * La siguiente función recibe un array de cadenas y devuelve un
 * array de funciones. Cada función del array resultante imprime un
 * mensaje que contiene el nombre del elemento correspondiente en el
 * array de entrada.
 */
function saludadores(nombres) {
    var result = new Array(nombres.length);
    for (var i = 0; i < nombres.length; i++) {
        result[i] = () => { console.log(`¡Hola, ${nombres[i]}!`); };
    }
    return result;
}

var fs = saludadores(["Diana", "David", "Darío"]);
fs[0]();
fs[1]();
fs[2]();
```

Ejecuta este fragmento de código. ¿Qué resultado obtienes? ¿Coincide con el esperado? Modifica el programa para que la función `saludadores` se comporte de la manera especificada en el enunciado.

▷ **4.** Implementa las siguientes funciones utilizando exclusivamente funciones de orden superior. ¡No está permitido recorrer los arrays mediante bucles!

(a). Escribe una función `pluck(objects, fieldName)` que devuelva el atributo de nombre `fieldName` de cada uno de los objetos contenidos en el array `objects` de entrada. Se devolverá un array con los valores correspondientes. Por ejemplo:

```
var personas = [
    {nombre: "Ricardo", edad: 63},
    {nombre: "Paco", edad: 55},
    {nombre: "Enrique", edad: 32},
    {nombre: "Adrián", edad: 34}
];

pluck(personas, "nombre") // Devuelve: ["Ricardo", "Paco", "Enrique", "Adrián"]
pluck(personas, "edad")   // Devuelve: [63, 55, 32, 34]
```

(b). Implementa una función `partition(array, p)` que devuelva un array con dos arrays. El primero contendrá los elementos `x` de `array` tales que `p(x)` devuelve `true`. Los restantes elementos se añadirán al segundo array. Por ejemplo:

```
partition(personas, pers => pers.edad >= 60)
// Devuelve:
//  [
//      [ {nombre: "Ricardo", edad: 63} ],
//      [ {nombre: "Paco", edad: 55}, {nombre: "Enrique", edad: 32},
//        {nombre: "Adrián", edad: 34} ]
//  ]
```

(c). Implementa una función `groupBy(array, f)` que reciba un `array`, una *función clasificadora* `f`, y reparta los elementos del `array` de entrada en distintos arrays, de modo que dos elementos pertenecerán al mismo array si la función clasificadora devuelve el mismo valor para ellos. Al final se obtendrá un objeto cuyos atributos son los distintos valores que ha devuelto la función clasificadora, cada uno de ellos asociado a su array correspondiente. Ejemplo:

```
groupBy(["Mario", "Elvira", "María", "Estela", "Fernando"],
        str => str[0])         // Agrupamos por el primer carácter

// Devuelve el objeto:
// { "M" : ["Mario", "María"], "E" : ["Elvira", "Estela"], "F" : ["Fernando"] }
```

(d). Escribe una función `where(array, modelo)` que reciba un `array` de objetos y un objeto `modelo`. La función ha de devolver aquellos objetos del array que contengan todos los atributos contenidos en `modelo` con los mismos valores. Ejemplo:

```
where(personas, { edad: 55 })
    // devuelve [ { nombre: 'Paco', edad: 55 } ]
```

```
where(personas, { nombre: "Adrián" })
    // devuelve [ { nombre: 'Adrián', edad: 34 } ]

where(personas, { nombre: "Adrián", edad: 21 })
    // devuelve []
```

▷ **5.** Escribe una función `mapFilter(array, f)` que se comporte como `map`, pero descartando los elementos `obj` de la entrada tales que `f(obj)` devuelve `undefined`. Por ejemplo:

```
mapFilter(["23", "44", "das", "555", "21"],
            (str) => {
                let num = Number(str);
                if (!isNaN(num)) return num;
            })

// Devuelve: [23, 44, 555, 21]
```

▷ **6.** En la Figura 1 se muestra el diagrama UML correspondiente a tres clases:

- Los objetos de la clase `Figura` representan figuras geométricas. Cada uno de ellos tiene asignada una posición (coordenadas `x` e `y`) y un color. Todas las propiedades son de lectura y de escritura, pero solamente se permitirá escribir en el atributo `color` si el valor escrito es una cadena que contenga un color HTML en notación hexadecimal (por ejemplo, `"#23B5DD"`). El valor por defecto del color es `"#000000"`.

  El método `esBlanca` devuelve `true` cuando se llama sobre un objeto cuyo `color` es `"#FFFFFF"`. El método `pintar` imprime el siguiente texto por pantalla,

  ```
  Nos movemos a la posición ([x], [y])
  Cogemos la pintura de color [color]
  ```

  sustituyendo `[x]`, `[y]` y `[color]` por sus respectivos valores.

- Los objetos `Elipse` heredan de `Figura` y contienen dos atributos adicionales: `rh` (radio horizontal) y `rv` (radio vertical). El método `pintar` debe imprimir lo siguiente:

  ```
  Nos movemos a la posición ([x], [y])
  Cogemos la pintura de color [color]
  Pintamos elipse de radios [rh] y [rv]
  ```

- Por último, los objetos `Circulo` representan elipses en las que `rh = rv`.

Representa esta jerarquía de clases en Javascript.

▷ **7.** Ejecuta el siguiente programa escrito en Javascript:

```
"use strict";

var persona = {
    nombre: "Gloria",
    conocidos: ["Alejandra", "Fran"],
    saludarAConocidos: function() {
        this.conocidos.forEach(function(conocido) {
```

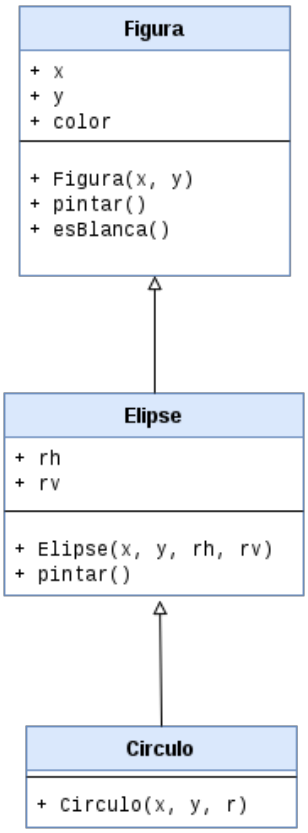

Figura 1: Diagrama de clases sobre figuras geométricas.

```
            console.log(`${this.nombre} saluda a ${conocido}`);
        })
    }
};
persona.saludarAConocidos();
```

¿Es correcta esta solución? ¿Por qué? ¿Qué pasaría si cambiases el argumento de la función `forEach` por una expresión lambda en lugar de utilizar `function`?

▷ **8.** Utilizando expresiones regulares, implementa las siguientes funciones:

(a). `esIP(str)`, que indique si la cadena dada representa una dirección IP válida.

(b). `esCorreo(str)`, que indique si la cadena dada representa una dirección de correo electrónico válida.

(c). `esPassword(str)`, que indique si la cadena dada tiene longitud entre 6 y 15 caracteres, y contiene, al menos, una letra, un dígito y un símbolo de los siguientes: `#`, `%`, `$`.

(d). `interpretarColor(str)`, que, dada una cadena que representa un color en formato `#RRVVAA`, devuelva un objeto con tres atributos (`rojo`, `verde` y `azul`) con el valor (en base 10) de la componente correspondiente. Si la cadena de entrada no es un color HTML válido, se devuelve `null`.

Indicación: utiliza `parseInt`.

(e). `numeroPalabras(str)`, que cuente el número de palabras contenidas en la cadena `str`.