Universidad Nacional de Costa Rica (UNA)
Escuela de Informática.
EIF-400 II-2017. Dr. Carlos Loría Sáenz (colaboración inicial de J.P. Arias)

Práctica General de FP usando ES6/7

Sección I

1. Para investigar: Indique año de creación, paradigma, estático ó dinamico de los siguientes lenguajes de programación (LP): FORTRAN, COBOL, LISP, PROLOG, PASCAL, C, SIMULA, ERLANG, OBJECTIVE-C, C++, JAVA, JAVASCRIPT, PYTHON, SCALA, SWIFT, KOTLIN, GO, ELIXIR.

2. Revise cada PPT (hasta el slide que haya sido cubierto en cada uno) buscando preguntas y ejercicios ahí planteados y resuélvalos. Lo mismo con ejercicios no resueltos disponibles en el sitio.

3. Explique el rol que juegan los combinadores de `FP` con relación a la programación imperativa.

4. Usando `HTML`, `CSS`, `JS` como ejemplos explique programación declarativa versus imperativa. Dé ejemplos apropiados.

5. Explique por qué las clausuras de `Java` no permiten reasignar variables que no son locales.

6. Demuestre con un ejemplo que un programa en Java se puede caer por un `ClassCastException` aunque si haya pasado el análisis de semántica estática. ¿Podría un compilador evitarlo en todos los casos? Sugerencia: Use el problema de parada de una Máquina de Turing (`halting problem`)

7. Considere la siguiente función que calcula simultáneamente el promedio entre el máximo y mínimo de un arreglo no vacío de números. Reescriba usando combinadores. Mueva donde corresponda a ES6.

```
function maxMin(a){
  var m = {min:a[0], max:a[0]};
  for(var i = 1 ; i < a.length; i++){
      if (a[i] < m.min) m.min = a[i];
      if (a[i] > m.max) m.max = a[i];

  }
  return (m.max + m.min)/2;
}
```

8. Escriba una función `contains(x, l, f)` que retorne `true` si `l` contiene a `x`, `false` si no. El parámetro f es opcional se usaría para chequear igualdad. Si no se indica es las comparación por ==. Haga dos versiones. Una recursiva de cola y otra usando `Array.reduce`.

9. Implemente el `mergeSort` de manera funcional usando recursión

10. Defina una función `copy(x, n)` cuyo resultado sea una lista con `n` copias de `x`. Haga una versión recursiva y una que use un combinador.

11. Si $l=[l_1,\ldots, l_n]$ y $m=[m_1,\ldots, m_n]$ el producto interno es $l_1*m_1 + l_2*m_2 + \ldots + l_n*m_n$. Si ambas están vacías el producto es 0. Defina una función que calcule el producto interno de dos listas de dobles. Implemente una vez con recursión y otra usando combinadores. Verifique que se puede hacer usando un solo combinador o dos combinadores en serie. ¿Qué ventajas puede tener una versión sobre la otra?

12. La función `split(l, pivot)` retorna una lista `r=[m, M]` donde `m` es la lista de los números en `l` que son menores o iguales que `pivot` y `M` los mayores. Impleméntela con `reduce`. Generalice al caso que `l` contenga objetos que se puedan comparar de alguna forma. Esa forma dependa del tipo de los objetos.

13. Defina una función `majority(l)` que encuentre la diferencia entre la cantidad de `true`s y `false`s en un array `l`. ¿Puede hacerlo con combinadores?

14. Implemente `frecuencias(l, g, n)` que recibe un array `l`, una función de agrupamiento `g` que asigna un grupo `i` entre 0 a `n` a cada objeto en `l` que clasifica en grupo `i`; y `n+1` sino lo logra clasificar. Retorna un arreglo `a` tal que `a[i]` cuenta la cantidad de objetos en el grupo `i`.

15. Demuestre que es posible expresar `map` en términos de `reduce`.

16. Use `Array.every` para definir una funcione `repeated(a, x, k, f)` determinar (`true` o `false`) si un array `a` contiene el mismo objeto `x` repetido `k` o más veces. Como antes `f` es (opcionalmente) un comparador por igualdad opcional. Su valor default es usar ==.

17. Implemente `gcd(n, m)` el máximo común divisor entre `m` y `n` de manera funcional usando recursión de cola.

18. Modele el siguiente escenario (en donde corresponda use `RegExp` y `FP`): Una persona tiene nombre (`String`), edad (`Number`) y teléfono (`String` de la forma "`0?[289]\d{2}-\d{4}-\d{4}`") -y una lista de contactos que son personas. Las personas en dicha lista puede ser amigas ó conocidas (disjuntos). Defina una función `contactosDe(p, criterio)` que retorne la lista de contactos de una persona `p` que satisfacen el criterio `c` que es una función Úsela para expresar los siguientes queries
    - Todos los amigos de una persona `p`.
    - Conocidos ó amigos de `p` cuyo teléfono es celular (empiezan con 8 ó 9)
    - Lista de nombres y teléfonos de contactos de `p` que no tienen a `p` en su lista de contactos

- El promedio de edad de los amigos de `p`.

19. Modifique el ejercicio anterior para que opcionalmente los resultados salgan ordenados según cada resultado.

20. Defina una función `pascal(n)` que retorna la lista de las `n` primeras filas (numerando desde 0) del triángulo de Pascal. Defina otra función `imprimaPascal(filas)` que imprima la salida de `pascal` (salida a consola). Ejemplo con `n=3`.

```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
```

Sección II

1. Defina una función `cuadrados` que tome un número natural `n` y retorne una lista con los primeros `n` números cuadrados.

   Ejemplo: `cuadrados(4) = [1, 4, 9, 16]`

Luego, obtenga la suma de los primeros 50 cuadrados

2. Escriba una función `cuantosPrimos(a)` que cuente cuantos números primos existen en una lista[i] de números `a`.

3. Escriba una función `diag(l)` que tenga una lista de hileras como parámetro y que imprima los caracteres en una diagonal.

Ejemplo: `diag (['a','b','c','d','e'])`

```
a
    b
        c
            d
                e
```

4. Escriba una función `aproxSeno` que, dados dos números `epsilon` y `x` (el primero mayor que 0, el segundo cualquiera), retorne el primer número `y` con la propiedad de que $|Math.sin(x) - y| < epsilon$. Use la aproximación de Taylor de seno (no use `Math` para calcular la función pedida).

5. Escriba una función `esSegmento`, que, dadas dos listas `xs` y `ys` devuelva `true` si `xs` es segmento de `ys`, y `false` si no.

6. Escriba una función `eliminarDuplicados`, que, dado un array, devuelva un array, con solamente una ocurrencia de cada elemento del array original.

7. Escriba una función recursiva `sc` (sublistas crecientes), que, dada una lista, devuelva una lista de listas que existan en todas las sublistas no decrecientes de la lista. Escriba también una definición de `sc` usando `reduce`.

Por ejemplo: `sc([6,1,4,8]) = [ [ ], [6], [1], [1,4], [4], [1,4,8], [4,8], [1,8], [6,8], [8] ]`

8. Escriba una función `segcrec` (segmentos crecientes), que dada una lista, devuelva una lista de listas que cumpla con las siguientes condiciones:

a. La concatenación de los elementos en el resultado devuelve la lista original

b. Todos los elementos del resultado son listas no decrecientes y tampoco son nulas (vacías)

c. Por cada segmento no decreciente `ys` de la lista dada, existe un elemento en el resultado.

Ejemplo: `segcrec([1,3,5,2,9,0,6]) = [[1,3,5],[2,9],[0,6]]`

9. Defina una función recursiva `triangulo` que devuelva la suma de todos los números naturales entre su argumento y cero.

Ejemplo: `triangulo(6) = 0 + 1 + 2 + 3 + 4 + 5 + 6 = 10`

10. Defina una función `mapAlt` que tome como argumentos 2 funciones y una lista, y aplique la primera función a las posiciones pares de la lista, y la segunda a los elementos en posición impar. Luego investigue sobre *Array.prototype*, y prototipos en general en Javascript, y como usarlos para poder llamar a `mapAlt`, de la misma manera que se llama a `map`, en vez de pasar la lista como argumento.

11. El *Algoritmo de Luhn* es una forma simplificada de validar números de tarjetas de crédito calculado un "checksum" de sus dígitos. El algoritmo está especificado así:
    i. Considere cada dígito como un número separado.
    ii. Duplique de número por medio empezando del penúltimo número hacia atrás. por ejemplo: 1 **2** 3 **4** 5 **6** 7
    iii. Reste 9 de cada número que sea mayor a 9.
    iv. Sume todos los números
    v. Si el resultado de esta suma es divisible por 10, la tarjeta es válida.

    Visualice el número de tarjeta como un arreglo, y utilice la función creada en el ejercicio anterior para iterar sobre éste.

12. Escriba una función `multiplicar`, que, dados un número positivo `b` menor que 10 y una lista de números `ns`, que representa un número `n` en base `b`, devuelva una lista que represente la multiplicación `n*m`, según la representación descrita en el

ejercicio anterior. Puede inicialmente suponer que trabajamos en un sistema de numeración en base 10. Ejemplos:

a. `multiplicar(3, [9,8,4])= [2,9,5,2]`

b. `multiplicar(5, [6,4,1,9,9,9])= [3,2,0,9,9,9,5])`

Piense en el sistema tradicional de multiplicación con acarreo para resolver este problema.

13. Escriba una función `unico`, que, dada una lista devuelva una lista que contenga exactamente los elementos que se encuentran solamente una vez en la lista dada. Por ejemplo:

```
unico([a,b,a,h,v,b,t,g]) = [b, h, v, t, g]
```

14. Escriba una función recursiva `numDifAsc` que, dada una lista de números ordenada en forma ascendente, calcule cuantos elementos diferentes están en la lista. ¿Cuál es el tiempo de corrida de su función? ¿Qué pasa si no está ordenada?

Por ejemplo: `numDifAsc([1,2,2,5,5,5,5,7,8]) = 5`

15. Haga el ejercicio anterior usando combinadores.

16. Defina una función recursiva `lcs` (listas con suma), que dado un número natural positivo `s`, devuelva una lista de listas que cumpla con las siguientes condiciones:

a. Todos los elementos son crecientes y ninguno de los elementos contiene un número repetido

b. La suma de los números en cada elemento es exactamente el número dado

Por ejemplo:

```
lcs(6) = [[1,2,3], [1,5], [2,4], [6]]

lcs(8) = [[1,2,5], [1,3,4], [1,7], [2,6], [3,5], [8]]
```

17. Escriba una función recursiva `menosUno`, que calcule todas las sublistas de una lista que se forman a partir de una lista original eliminando un elemento.

Por ejemplo:

```
menosUno([1,2,3,4,5]) = [[2,3,4,5],[1,3,4,5],[1,2,4,5],
[1,2,3,5],[1,2,3,4]]
```

Ahora redefina utilizando combinadores.

Sección III

1. Mueva el siguiente código (manualmente) a promesas de forma que el caso funcione

eif400.examen.js

```
1
2  const fs = require('fs')
3
4  const readFileWithFS = (filename, handleData, handleError) =>
5      fs.readFile(filename, (error, data) => (error) ? handleError(error) : handleData(data))
6  ;
7  function testReadFileWithFS(filename){
8      console.log(`>>> Reading ${filename} <<<`);
9      readFileWithFS(filename, data => console.log(`*** ${filename} bytes read: ${data.length} ***`)
10                                  , error => console.error(error))
11     console.log(`>>> Reading was deferred <<<`);
12 };
13
14 testReadFileWithFS('eif400.examen.js');
```

Versión promisificada

```
function readFileWithFSPromise(filename){ // Hacer!!
}

function testReadFileWithFSPromise(filename){
    console.log(`>>> Reading ${filename} <<<`);
    readFileWithFSPromise(filename)
      .then (data => console.log(`*** ${filename} bytes read: ${data.length} ***`))
      .catch(error => console.error(error));
    console.log(`>>> Reading was deferred <<<`);
}
testReadFileWithFSPromise('eif400.examen.js');
```

2. Logre el siguiente caso de uso: Defina `RNumber` en `eif400.number.js`

```
class RNumber extends Number {

module.exports = {
      RNumber
}
```

Lo siguiente debe funcionar

C:\Windows\System32\cmd.exe - node

```
PP:node
> var {RNumber} = require('./eif400.number')
undefined
> for( v of new RNumber(5) ) console.log(v)
0
1
2
3
4
undefined
> n = new RNumber(665)
{ [Number: 665] val: 665, actual: 0 }
> n + 1
666
>
```

[i] Lista se entiende como arreglo en este contexto