Universidad Nacional de Costa Rica (UNA)
Escuela de Informática
EIF400 II-2023 Paradigmas de Programación
Dr. Carlos Loría-Sáenz

Recopilación de Correos con Ejercicios Al 9/10/2023

Contenido

Ejercicios 1a

1) Repase estos conceptos

- Abstracción (computacional)
- Programación imperativa
- Programación estructurada
- Manejo automático de memoria (recolección de basura)
- OOP pur versus impuro
- Concurrencia con multicore

2)
Asuma un assembler con estas especificaciones (por ahora). Digámosle por broma `ASSONE` (el `ONE` es por UNA).

Hardware:
Un CPU con memoria RAM, una consola con su teclado para entrada/salida.

Sintaxis
Similar a un ensamblador como el que Ud. conoce, un programa es una secuencia de intrucciones. Una instrucción por línea.
Una instrucción empieza opcionalmente con una etiqueta (identificador seguido de :), el nemónico y los argumentos
Comentarios de una línea con ; (punto y coma)
Ejemplo

```
SUMA: ADD AX BX ; Calcula AX += BX
```

Registros Generales de aritmética
`AX, BX`
Registro de dirección de memoria
`BP`
Memoria `M`
Un arreglo desde 0 hasta un máximo de posiciones que guardan números binarios
Se direcciona usando `BP` o una constante, por ejemplo: `M[0]` o `M[BP]`
Entrada/Salida
La posición 0 de `M` está reservada para imprimir, es decir, lo que esta en `M[0]` será impreso en decimal en consola si se ejecuta `DIS` (display).
Y si se ejecuta `INP` se lee de consola un número decimal y se pone en `M[0]` su versión en binario
Operaciones aritméticas
`ADD, MUL, SUB, DIV` toman `AX` y `BX` hacen la operación y dejan el resultado en `AX`
`INC DEC` : incrementan o decrementan en 1 el registro `AX` o `BX` o `BP`
Operaciones de movimiento
`MOV`: mueve de memoria a registro y de registro o contante a memoria.
Por ejemplo

```
MOV AX 0 ;pone un cero en AX.
MOV M[1] BX ;pone el contenido de BX en M[1]
```

`DIS`: escribe en consola en decimal lo que haya en `M[0]` (pasa de binario a decimal)
`INP`: pone en `M[0]` lo que se teclee en consola (en decimal y se convierte a binario)

Problema
Escriba un programa en ASSONE que lea tres números `a, b y c` de consola e imprima en consola el resultado `b ** 2 - 4*a*c`

3) Use un prompt como este en ChatGPT o similar (puede refinarlo con subsecuentes preguntas para mejor contexto)

"Dame un ejemplo que me permita entender go rutinas en el lenguaje Go"

Estudie las respuestas y sea capaz de explicarlas.

Ejercicios 2a

Repase, revise enriquezca estos conceptos mencionados en clase

- Expresión versus estatuto. Reflexione si en Java haya una expresión que también sea estatuto. Spoiler alert ...
- Virtual machine y WORA
- JVM
- Bytecode y .class
- JIT compiler (versus AOT compiler. Investigar este último).
- Stack Machine (ventaja sobre una máuina de registros)
- AST
- Postorder y generación de bytecode
- Desensamblar código (alias `javap`)

2) Tema a investigar (posible extra) y formarse una idea sobre GraalVM (Nota: ChatGPT3.5 puede estar desactualizado sobre el tema)

3) Observé que varios de Uds. no tienen muy buenas prácticas de estilo de código y ya son programadores avanzados. Les recomiendo estudiar algunas, por ejemplo, acá las de Google. Invierta en eso, es por su propio beneficio y eso es parte de una buena ingeniería de software!

4) [Requiere tiempo y mucha paciencia. Es una variante de lo que hicimos en clase] Usando Java y programación imperativa como la que Ud. conoce: escriba un método recursivo `static int factorial(int n)` que retorne el factorial de `n`. Use operador ternario para que sea una sola expresión.

a) Haga el AST de la expresión retornada, genere el post-order. Cuando sea un llamado a una función asuma que hay un nodo `INVOKE` (también se le puede llamar `CALL`) cuyos hijos los argumentos de izquierda a derecha (tantos hijos como argumentos haya) seguido del método llamado. Es decir si hay una expresión `foo(a, b)` habría un nodo en el AST `INVOKE` con 3 hijos de izquierda a derecha `a`, `b`, `foo`.

b) Use `javap` y vea el bytecode generado y compare con su versión en post-orden
Nota: Para saber sobre la lista de intrucciones del bytecode de la JVM puede ver acá. Recuerde siempre que es una stack machine.

Ejercicios 2b

1) Repase y extienda estos conceptos

- Lenguaje: Faceta declarativa versus operativa
- Definición de clase versus instancia
- Método de clase versus de instancia
- Tipos estáticos y su función

- Paquete (`package`) en Java
- Modificadores de declaraciones (`public`, `private`, `protected` y ninguno. (Investigar los no vistos en clase).
- Parámetros formales versus reales
- Compilación y AST de un operador ternario
- Desactivación del JIT y efectos

2) Haga ejemplos y compile para tratar de contestar las siguientes preguntas
a) Sea `C` (en mayúscula) una `class` de Java y `m` y método estático en `C` asuma por simpleza de tipo `void` y sin parámetros formales. Sea `c` (minúscula) una instacia de `C`. ¿Sería válido: `c.m()`?

b) Sea `B` que extiende a `C`. ¿Podría `B` tener un método estático `void m()` como lo tiene `C`?

c) ¿Podría haber declaraciones de una `A` tal que un bloque de código Java así que compilara correctamente?

```
A a = new A();
a.foo();
a.foo(a);
a.foo(a, a, a);
A[] b = new A[10];
a.foo(b);
```

3) Reescriba la función `int fact(int n)` de forma que no use un ternario sino un **`if-then-else`** estatuto. ¿Cambiará eso el código generado por el compilador de Java?

4) ¿Dibuje el AST de `h = Math.sqrt(a * a + - b * b)`? Asuma que = es un operador. Piense cómo debería armarse el ábol de forma que un post-order sea correcto al ejecutarlo en una stack machine?

5) Suponga que hay una clase `A` declarada en un paquete `a.b.c`. Asuma que `A.java` está en una carpeta `src` y se quiere que javac deje su compilación en una carpeta `classes`. Asuma que `A` tiene `main`. Ambos `src` y `classes` deben estar en una carpeta `project`.
a) ¿Qué habría que teclear en consola para poder compilar (según se pide) `A`?
b) ¿Qué habría que teclear en consola para poder ejecutar `A`?
c) ¿Qué habría que teclear para desensamblar `A`?

Ejercicios 4a

Nota: En el grupo de las 6pm no hemos llegado a cubrir todo. Pero, qué tal si lo intenta igualmente. ;-)

1) Revise, entienda y amplie los siguientes conceptos/afirmaciones

- Partes de un compilador
- Parser (Syntax Analysis)
- Typer (Static Semnatic Analysis)
- AST
- ST (symbol table)
- Syntax Error
- Static Semantic Error
- Lenguage "estático" vs "dinámico"
- Explique afirmación: "*Un lenguaje estático son dos lenguajes en uno*".
- Reference type vs Primitive Type
- Boxing/Unboxing
- Type parameter
- Generics y su beneficio
- Type erasure (borrado de tipos)

2)
a) Considere el siguiente código

```
var a = new Integer[]{1,2,3};
var b = new ArrayList<Integer>();
```

¿Cuál es el tipo estático del respectivo lado derechos de esas dos asignaciones?
¿Cuál es el tipo dinámico de `a` y cuál el de `b`? (puede averiguarlo usando `a.getClass()` y `b.getClass()`)
Compare esos tipos y explique una importante diferencia que no es la obvia: que uno es un `array` y el otro un `ArrayList`?

b) Explique de manera clara y precisa por qué el siguiente código no compila en Jshell (por ende ni con Javac)

```
List<Apple> la = new ArrayList<>();
List<Fruit> lf = la;
```

¿Cómo se podría corregir el problema?

c) Verifique que este ejemplo abajo **sí compila** en el `Jshell` (y con `javac`) (a pesar de que en principio polucionaría `aa`) pero la JVM sí detecta el problema con una excepción en tiempo de ejecución. Indique cómo se llama la `Excepción`

```
class Fruit{};
class Apple extends Fruit{}
class Orange extends Fruit{}
Apple[] aa = new Apple[1];
Fruit[] fa = aa;
fa[0] = new Orange();
```

3) Considere el siguiente código

```
final class Person{
    final private String name;
    final private int age;
    public String getName(){return this.name;}
    public int getAge(){ return this.age;}
    public String toString(){
```

```
        return String.format("Person(%s, %d)", name, age);
    }
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

Construya una **class** `PersonComparator{...}` que permita comparar (menor a mayor) dos instancias de `Person` primero por `name` y luego por `age`. Construya ejemplos de prueba que compare a los siguientes objetos. Use Comparator del JDK.

```
var juan1 = new Person("Juan", 20);
var juan2 = new Person("Juan", 25);
var maria = new Person("Maria", 20);

class PersonComparator ... // su respuesta

Comparator<Person> cp = ... //  su respuesta
println(cp.compare(juan1, juan1)) // son "iguales"
println(cp.compare(juan1, juan2)) // juan1 "es menor" que juan2
println(cp.compare(juan1, maria)) // juan1 "es menor" que maria
```

Ejercicios 4b

1) Revise, repase y extienda los siguientes conceptos:

- Valor
- Tipo
- Sistema de Tipos (Typer) como sistema formal de razonamiento
- Reference type vs Primitive type
- Operaciones y relaciones sobre y entre tipos (<: y :) y sus propiedades de reflexividad y transitividad
- Generics y varianza (in-, co- y contra-) y casos de uso en Java
- Narrowing de tipos (no lo hemos visto a las 6pm aún)

2) Explique por qué `int x = null;` no compila pero `Integer y = null;` sí (equivale a preguntarse de qué lado de las dos castas está `null`)

3) Queremos un modelo de datos de una clase de objeto `Point` que modela un par `(x, y)` donde `x`, `y` pueden ser de cualesquiera tipos que sean comparables (en el sentido de `Comparable` de Java). Les diremos a `x`, `y` las coordenadas.
Un `Point` a la vez es comparable.
Dos `Points` `(x1, y1)` y `(x2, y2)` son "equal" si lo son entrada a entrada.
Para determinar si un `Point (x1, y1)` es "menor" que otro `Point (x2, y2)` se comparan lexicográficamente: si `x1` es "menor" que `x2` se cumple (independiente de los `y`), si `x1` es "mayor" que `x2` ya no se cumple del todo. Si `x1` es equal a `x2` se comparan

los `y` para decidir (`y1` deber ser "menor" o equal que `y2`). Podrían haber especializaciones de Point, por ejemplo PointInteger o PointString (en ambos casos las coordenadas siendo del mismo tipo, y todo lo que funcionaba bien)

4) Explique paso a paso si el siguiente código compilaría en cada caso, por qué sí o no, y las diferencias entre `copyA` hasta `copyG` sobre qué tipo de varianza se está usando.
Note que en un caso se usa un **método parametrizado** (que no hemos visto, pero son similares a tipos como `class<T>{}`), el(los) tipo(s) que se declara(n) justo antes del tipo de retorno.
En los casos que sí compile, cree para ese caso listas ejemplos de `a` y `b` (no nulas ni vacías) tales que el llamado a `copyX(a, b)` sí compila donde `X` es `A, B`, hasta `G` y `copyX` si compila.

```
void copyA(List<Integer> a, List<Integer> b){
    for (var e : a)
        b.add(e);
}

void copyB(List<Integer> a, List<? extends Integer> b){
    for (var e : a)
        b.add(e);
}

void copyC(List<Integer> a, List<? super Integer> b){
    for (var e : a)
        b.add(e);
}

void copyD(List<? super Integer> a, List<Integer> b){
    for (var e : a)
        b.add(e);
}

void copyE(List<? super Integer> a, List<? super Integer> b){
    for (var e : a)
        b.add(e);
}

void copyF(List<? extends Integer> a, List<? super Integer> b){
    for (var e : a)
        b.add(e);
}

<T> void copyG(List<T> a, List<T> b){
    for (var e : a)
        b.add(e);
}
```

5) Pida a ChaGPT una lista de 15 digamos 'pifias' (comportamientos peculiares) de JS en cuanto al manejo de tipos (conversiones implícitas).
Por ejemplo:
¿Qué retorna `666 + []`?
¿O cuánto es `NaN == NaN`?
Fórmese una opinión "educada" sobre JS (recuerde está entre los 3,-5 más usados del mundo :-)).

Ejercicios 5a

1) Revise, repase y extienda los siguientes conceptos

- Transpilación:
  - React JSX
  - TS
- Herramientas
  - `tsc`
  - `ts-node`
  - `nodemon`
- Programación Funcional (acortaremos a `FP`)
- Nivel de abstracción en FP sobre programación imperativa clásica (for, while, etc)
- Programación "a la Knuth" (como en programación literaria)
- Función como objeto, dato o valor (en inglés se dice *function as first-class citizen*)

2) En estilo FP y en JS (combinación que llamaremos FP-JS) y desarrollando casos de prueba que corran en `node`. Ponga sus respuestas en un script `semana5a.js` que se pueda correr así: `node semana5a.js`. Al menos dos casos de prueba por ejercicio.
Siga y complete el modelo del script. Ajuste los casos de prueba según cada ejercicio.
Busque un estilo Knuth (parafraseando: el arte de escribir código pensando empáticamente en el pobre diablo que va a tener que leerlo)
a) Escriba `compose(f, g)` tal que `compose(f, g)(x) = f(g(x))` para todo `x` talque `f(g(x))` esté definida.
b) Escriba `minOfArray(a)` que retorne el menor número en el array `a` de números, asumiendo que `a` no está vacío.
c) Escriba una función `lessThan(a, x)` que cuente cuántos elementos en el array `a` de números son menores que el número `x`
d) Escriba una función `split(a, f)` que reciba un array `a` y una función `f` definida sobre los objetos en `a` y que puede retornar `true` o `false`, tal que se cumpla que `split(a, f)` retorna un array `[yes, no]` donde `yes` un array con elementos `e` en `a` que cumplen `f(e) = true` y `no` un array con el resto Que sea `O(n)` donde `n=a.length`. **Nota**: a cualquier función como `f` la llamaremos en FP un `predicado`.

Ejercicios 5b

1) Revise, repase y extienda

- Composición de funciones
- "Burocracia del diseño" en OOP (término no estándar que invento yo)

Para este punto use ChatGPT por ejemplo con el prompt de abajo, estudie su respuesta, esté en disposición de explicarla.
Nota: Ud. puede no estar de acuerdo conmigo o con ChatGPT de que la burocracia del diseño sea mala per se. Sea crítico y asertivo.
Prompt
En programación orientación a objetos, en especial en lenguajes como Java, hay una alta, digamos, burocracia de diseño, quiero decir, se deben crear las clases y las relaciones entre ellas (como herencia) antes de poder escribir código operacional.

2) En matemática `o` es un operador sobre funciones (y relaciones). ¿Cuáles propiedades algebraicas cumple? (indique falso o verdadero, en el primer caso dé un contraejemplo) donde `f, g, h` son cualesquiera funciones que se puedan componer. Indique el nombre de la propiedad en cada caso.

a) `f o i = i o f = f` (donde `i` es la función identidad)
b) `f o (g o h) = (f o g) o h`
c) `f o g = g o f`

3) Escriba en FP-JS una función `multiple_comp(a)` que reciba un array de funciones `a` (asuma que todas son de un solo argumento), y retorna una función que se comporta como la composición de todas las funciones en `a`. (Nota: no le hemos dicho, pero recursión es considerado una parte natural de FP).

Reglas para `multiple_comp(a)`

1. Si `a` está vacía retorna la función identidad
2. Si `a` solo tiene una función retorna esa función
3. Si `a` tiene `n` funciones `f1, f2, ..., fn` con `n > 1` entonces retorna una función que se comporta como `f1 o f2 o ... o fn`

Nota: Usamos que `(f o g)(x) = g(f(x))` (`g` se come lo que `f` generó primero)
Ejemplo (úselo como caso de prueba)
Asuma `a = [f1, f2, f3]` tales que

```
f1(x) = 2 * x
f2(x) = x ** 2
f3(x) = x + 2
```

Si `h = multiple_comp(a)` entonces para un humano, que hace el trabajo algebraico de derivar la forma de `h`, se debería cumplir que

```
h(x) = multiple_comp(a) = (f1 o f2 o f3)(x) = 4 * x**2 + 2
```

Ejercicios 6a

Repase, revise, amplie estos conceptos

- (In)mutabilidad de datos: ventajas y limitaciones
- Efectos secundarios
- Transparencia referencial
- Estándares ESM (`import/export`) y CJS(`require`) de módulos en JS
- Desestructuración en JS

1) Lea un poco sobre por qué el lenguaje `Go` (aka Golang) usa una forma de OOP que no es la usual en lenguajes como C++ o Java. ¿Qué razones de diseño los llevaron a esa decisión en Google?

2) Si no lo hizo, haga `comp_multiple` de semana 5b usando `reduce` (debe salirle en una línea). Haga pruebas de que funciona.

3) Pruebe que el patrón (o combinador) de FP `filter` se puede simular con `reduce`
Es decir haga una función `filter(a, f)` que se comporte como `a.filter(f)` pero implementada con `a.reduce`.

4) En Dia-10 (acá) encuentra uan forma de un objeto literal que representa una página html codificada como objeto de JS (ver variable doc). Un elemento es de la forma `{tag, atts, children}` `tag` es el string del tag, `atts` un array de los atributos y `children` un array de los elementos debajo de ese tag. Los atributos son objetos de la forma `{name:value}`. Hay una excepción que son los elementos textuales, que son simplemente hileras.

Hints. El objecto `Object` tiene métodos para extraer propiedades de objetos. Por ejemplo: `Object.keys`, `Object.values`, `Object.items`.

Escriba una función `element_toString(element)` que reciba en `element` un objeto de una codificación así y devuelva una hilera que sea el html de la página codificada en JS.
Por ejemplo, para el valor `doc` que viene en el archivo, la salida sería una hilera como:

```
<html ><head ><title >My Page </title> </head> <body ><h1
style="color:blue;text-align:center" >Hello World </h1> <div
id="App" ></div> <script src="../scripts/main.js"
type="application/javascript" ></script> <script
type="application/javascript" >
let n = 666;
 console.log(n)
 </script> </body> </html>
```

Que "embellecida" se vería así (pero su programa **no necesita emebellecerla**, para eso hay herramientas aparte)

```
<html>
  <head>
    <title>My Page </title>
  </head>
  <body>
    <h1 style="color:blue;text-align:center">Hello World
</h1>
    <div id="App"></div>
    <script
src="../scripts/main.js"    type="application/javascript"></s
cript>
    <script type="application/javascript">
      let n = 666;
      console.log(n)
    </script>
  </body>
</html>
```

Se espera una solución FP (que aproveche desestructuración) en un módulo `html.js` (CJS) y un módulo `test_html.js` que usa `html.js` y genera la salida.
Puede modelar usando funciones auxiliares, para un mejor estilo Knuth.

Ejercicios 6b

0) Revise brevemente estas dos librerías que le mencioné en clases: `lodash` (es más que para FP) y `underscoreJS` (sí es más para FP). La última fue hecha antes de ES6, entonces algunas coas ya está en el API de Array.

1) Revise el API de Array en JS. Si llamamos "combinador" de arrays a cualquier método en ese API que reciba una función de parámetro (Nota: a una función de parámetro también se le dice una `callback function`, o `callback` simplemente).
Esté en capacidad de implementar usando FP su propia versión de cada uno de esos combinadores. Nota: que no sean las que ya implementamos como `filter` y `map`. **Nota**: los callbacks de `Array` reciben hasta `n + 2` parámetros `x1,...,xn, i, thisArg`, siendo `x1,`

`..., xn` los propios del callback, `i` es el índice del elemento que está siendo visto y `thisArg` el array (a menos que se cambie; lo cual veremos luego en clase)
Por ejemplo

```
let a = ['a', 'b', 'c']
a.forEach((e, i) => console.log(`${e+i})`))
```

Imprime

```
a0)
b1)
c2)
```

2) Justifique en cada caso abajo, cuál identidad es válida y cuál no. En este último caso, dé un contraejemplo
Sean `f` y `g` funciones cualesquiera tal que `f o g` existe y sea `a` un array cualquiera tal que los combinadores son aplicables a `a`.

```
a) a.map(f).map(g) = a.map(g).map(f)
b) a.filter(f).map(g) = a.map(g).filter(f)
c) a.map(f).map(g) = a.map(f o g)
d) a.filter(f).filter(g) = a.filter(g).filter(f)
```

3) Usando `reduce`, construya un combinador `zip` que cumpla `zip(a, b, f) = [ f(a[i], b[i]) / i=0,... ,n-1 ]` para cualesquiera array `a` y `b` tales que `n = a.length = b.length`. Razone sobre el tiempo de corrida y consumo de memoria

4) Implemente en FP `reverse(a)` que dado un array `a` retorna un array `b` tal que `b[i] = a[n-i-1]` para `i=0,...n-1` donde `n=a.length`

5) Sea `add_curry = x => y => x + y`
Sea `foo = x => add_curry(x)(x)`
Dado lo que hace `foo`, qué nombre es más apropiado en vez de `foo`. Justifique su respuesta.

6) Diga qué hace la función foo siguiente y póngale un nombre acorde a su propósito.

```
const foo = n => Array.from(new Array(n), (_, i) => i)
```

Ejercicios 7a

1) Revise, repase y profundice:

- Combinadores `call`, `apply`, `bind`
- Resolución de "binding" (RB)
- RB dinámico versus RB estático
- Asincronía versus sincronía
- Stack frame
- JS Event Loop

2) Considere la función `foo`

```
function foo(n){
    return n <= 1 ? n : foo(Math.floor(n / 2))
}
```

Asumiendo que `n` es entero: encuentre el O-grande del número de frames (`foo_frames(n)`) que ocupa `foo(n)` en pila.
Es decir, cuánto es `?` en `foo_frames(n) ~ O(?)`

3)
Considere el código abajo:
Nota:
Escribir en un objeto como `{..., f : function(){...}, ....}` en ES6 se puede escribir `{..., f(){...}, ....}` en notación de método.

```
const obj = {
    x:'x in obj',
    local_obj:{
        x:'x in local_obj',
        getA(){const local = function(){return this.x};
return local()},
        getB(){const local = () => this.x;  return local()},
        getC(){const local = ()
=> this.x;return  local.call(this)},
        getD(){return  local = () => this.x;}

    },
    getX(){return  this.x}

}
console.log("*** Cases of 'this' binding to explain ****")
console.log(`Case 1: obj.local_obj.getA() =
'${obj.local_obj.getA()}'`)

console.log(`Case 2: obj.local_obj.getB() =
'${obj.local_obj.getB()}'`)

console.log(`Case 3: obj.local_obj.getC() =
'${obj.local_obj.getC()}'`)

console.log(`Case 4: obj.local_obj.getD() =
'${obj.local_obj.getD()()}'`)

console.log(`Case 5: obj.getX.bind(this).call(obj.local_obj)
```

```
= '${obj.getX.bind(this).call(obj.local_obj)}'`)
```

Explique la salida y justificando el porqué de la misma, la que se obtiene en cada caso.

Ejercicios 7b

1) Revise, repase y extienda

- Arquitectura básica de JS (stack, queue, heap)
- Eventloop (y libuv)
- Timers
- I/O asincrónico con fs
- Callback hell
- Explicar gráficamente la salida de un código sincrónico vs asincrónico

Nota: Como alguien me hizo ver algunos también mencionan el `Promise hell` para favorecer `async/await`. ¡Cierto!

2) Escriba un combinador `repeatUntilWhen({state, what, when, then, ms})` que de manera asincrónica (non-blocking) ejecute `what(state)` cada `ms` millisegundos hasta que `when(state)` sea verdadero. Al terminar se ejecuta `then(state)`. Por defecto, si no viene, `then` es la función identidad y `ms` es 1000 (Investigue `ES6 default parameters`). No use promesas ni async/await.

```
// Test case
const state = {x: 0}
const  what = state => state.x += 1;
const  when = state => {
console.log(`When: current x=${state.x}`); return state.x > 3;}
const  then = state => console.log(`Final x = ${state.x}`)
repeatUntilWhen( {state, what, when, then, ms:1000} )
```

Salida (cada segundo)

```
When: current x=0
When: current x=1
When: current x=2
When: current x=3
When: current x=4
Final x = 4
```

3) Revise el demo explicado hoy. Añada tres keywords más (class, const, of) y recompile y pruebe que todo funciona. Añada un style `TextArea.css` que cambie un poco el CSS del componente `TextArea` (a su gusto)

Ejercicios 8a

1) Revise, repase y extienda:

- React virtual dom y árbol de componentes
- Concepto de estado y su rol en rendering
- Estado por "props", estado local (stateless vs statefull)
- Componentización por herencia (OOP) versus por composición (FP)
- Hooks y ejemplos de uso básicos: `useState`, `useEffect`, `useRef`, `useCallback`
- Build de un proyecto de create-react-app y corrida usando un server como http-request.

2) Tome el demo "text_select" y añada un nuevo componente nuevo SelectorButtom que se comporta como un (toggle) botón que al darle click impide que el select apareza aunque se dé Escape. Al darle click reactiva la aparición usual del select. Añádale algo de forma que se aprecie si el botón está impidiendo o permitiendo que se vea el select. Estado inicial es permitiendo.

3) Prepare para el jueves 21/9 un pequeño informe (2-3 páginas) impreso sobre el estado de avance de OneFlow en P1 a la fecha. Vale por dos quices grupales. Me lo entrega cada coordinador en los primeros minutos y luego me enseñan un poco de lo que tienen. No acepto hojas sueltas, por favor.

Para ese reporte, lleve un **documento impreso**, siguiendo este modelo:

```
Título: EIF400 II-2023 Informe I Avance OneFlow P1
Fecha: 21/9/2023
Grupo: GG-HH (código de grupo)
Nota: (deje espacio a poner por el profesor)
Observaciones: (deje espacio de unas 4 líneas a poner por el profesor)
Lista de Miembros (pero lo firma el(la) coordinadora)
I. Sobre Trabajo grupal
Forma de Organización: (añada una descripción de cómo se han
organizado, periodicidad de coordinación y control de avance). Indique
una nota autoevaluativa  (0-100) sobre este punto.
Puede incluir observaciones relevantes. (Máximo una página).
II Sobre "features"
Tome como referencia el correo "Sobre features de OneFlow P1" del día
16/9 y genera un check list donde a cada punto le añade un número
entre 0 y 100 sobre avance estimado.
Promedio: Ponga acá el promedio de la autoevaluación de cada feature.
Puede incluir obervaciones relevantes.
```

Evaluación del Informe:
I Sobre Trabajo grupal (50%) Apreciación subjetiva a criterio del docente.
II Sobre "features" (promedio de autoevaluación de cada punto escalado por nota grupal) (50%).

Ejercicios 9a

0) Revise, repase y extienda

- `useReducer` React hook
- **`export default`**
- Desestructuración y el spread operator
- El objeto `Promise` tiene varios combinadores que Ud. debe conocer: `all`, `any`, `allSettled`, `race`. Investigue su función y haga ejemplos ilustrativos.
- No visto aún: Investigue si no sabe qué se entiende por `CORS` en programación Web, pues lo va a necesitar cuando su server hable con el server en Prolog, pero además es importante entederlo en general.

1) Escriba una función `secondPlusThirdOfArray(a)` que reciba un array de al menos 3 números `a` y retorne la suma del segundo más el tercer elemento. No puede usar el acceso usual con `a[...]`.

2) Modifique el demo de reducer explicado hoy de forma que si se teclea palabras (`typedText`) pone se muestran en mayúscula la primera letra de cada palabra (`toShowText`). Use FP.

3) Haga el ejercicio 2 de la `semana 7b` si no lo ha hecho

4) Modifique el ejemplo `readfile.js` desarrollado hoy en clase que maneje el `err`.

Ejercicios 10a

0) Revise, repase y amplíe:

- Metamodelo de objetos
- Relaciones en el grafo de prototipos
- Own properties vs metaproperties
- `__proto__` versus `prototype`
- Explique la afirmación: "__proto__ permiten la herencia en JS"
- `constructor`
- `Function` versus `Object`

0) **Sugerencia**: haga un documento que recoja todos los ejercicios asignados tras cada clase y haga soluciones.

1) Escriba una funcion `allConstructorsUpTo(obj)` que recibe un objeto `obj` de JS y retorne la lista de los constructores que se encadenan partiendo de `obj.constructor`. Asuma que `obj` no es `null` ni `undefined`. No puede usar programación imperativa.

2) Usando ES6, haga un modelo OOP que modele un árbol binario `BinaryTree` y árbol binario de búsqueda `BinarySearchTree` que hereda de `BinaryTree`. Encapsule en un módulo ESM `tree.mjs`. Haga pruebas de cada método en árboles no triviales en `testTree.msj`.

Requerimientos mínimos:
* `BinaryTree` tiene estos métodos para todo `BinaryTree t`
a) `t.left(), t.right(), t.root()` que retornan respectivamente lo esperado.
b) `t.isEmpty()` que retorna `true` si `t` está vacío, `false` en caso contrario
c) `t.nodeData(), t.isLeaf() t.isInterior()` predicados que retornan respectivamente lo esperado visto `t` como un nodo.
d) `t.height()` que calcula la altura de `t`. Un árbol vacío y las hojas de un árbol tienen altura cero por definción.
e) `t.balanceFactor()` que retorna la diferencia entre la altura del hijo izquierdo menos la del hijo derecho. Las hojas tienen `balanceFactor` de 0 al igual que un árbol vacío.

* `BinarySearchTree` tiene para todo `BinarySearchTree t`
a) `t.search(d)` retorna `n` si `n.nodeData() === d` en algún nodo `n` en el `BinarySearchTree t`. Devuelve `null` en caso de no existir `n`.

Ejercicios 10b

1 Repase, revise y extienda:

- ES6 **`class`** versus ES5 **`function`** (constructor)
- Patron Iterable y sus beneficios
- Definición de Iterable e Iterator en ES6
- Estructuras y algoritnos Lazy versus Eager
- `Symbol` y su objetivo
- `Symbol.iterator`

2) Verifique todo `string` en JS es iterable.

3) Modifique el `Nats` hecho hoy para que aparte del natural inicial (`init`), reciba, opcionalmente, un maximo (`max`) y un incremento (`increment`). Por ejemplo lo siguiente abajo debería funcionar (note que el `for` "por dentro automáticamente llama al iterador y hace que `n` sea el `obj.value` del objeto `obj` retornado por el iterador que este retorna y se detiene cuando `obj.done == true`. Similarmente con spread). No use (aún) generadores que no hemos visto:

```
> for ( const n of new Nats(5, 10, 2) ) console.log(n);
5
7
9
```

```
> a = [...Nats(5, 10, 2) ]
> console.log(a)
[ 5, 7, 9 ]
```

Sería como un range de python.

4) Construya `Digits` tal que algo como lo siguiente funcione (no use aún generadores para practicar):

```
> const digits = new Digits(66601)
> for (const d of digits) console.log(d)
6
6
6
0
1
```

5) Vaya preparandose para este caso de uso para OFS P1.5 (donde la parte de compile es aún "fake", Ud. será el compilador).

En `EA` cargamos desde el server, un script llamado, digamos, `ofs_test.ofs` así:

```
// script ofs_test.ofs
const nats = [* 0, n -> n + 1] // Es como nats = new Nats(0)
const even = nats >> [? n -> n % 2 == 0]  // es un pipe que
filtra números pares de nats produciendo un nuevo iterable
con ellos
const evenLessThanEleven = even >> [? n -> n < 11]  // pipe
que filtra del pipe even los pares menores que 11
evenLessThanEleven >> [> n -> console.log(n) ] // pipe que
imprime los pares menores que 11 del paso anterior
```

Note:
En OFS usaremos -> en vez de => de JS
>> es un operador (combinador) de composición de pipes que conecta un iterable de entrada para producir otro iterable de salida.
Ademas:
`[* ... ]` (se llama iterate)
`[? ...]` es el filter de iterables
`[> ...]` es el map de iterables
(después tendremos otros combinadores)

Ud a mano, escribe un módulo y lo pone en el server como `ofs_test.js` que tiene un código que usando iterables equivaldría funcionalmente al código en OFS que está en EA.

Es decir, Ud. es el compilador de OFS a JS solo para este ejemplo. (El jueves veremos formas de hacer esta "compilación a mano").

Cuando demos `/compile` se regresará simplemente en `TA` ese `ofs_test.js`.

Cuando seguido demos `/eval` en el server, se evalúará el `ofs_test.js en TA` y se capturará la salida de esa evaluación (¡investigue cómo hacerlo en node!) y esa salida se mostrará de regreso en RA.

Ejercicios 11a

1) Revise, repase, amplie:

- Iterable vs Iterator vs Generator
- Explique la frase "*un generator es un iterator sobre un código en vez de sobre una estructura de datos que es su iterable*"

2) Usando el `Nats` hecho en clase, reescriba a) el siguiente código usando un `for` (pero sin usar un índice como `i`) y b) lo mismo pero usando un `do while`

```
const nats5 = new Nats(5);
for(const n of nats5){
     if ( n > 11 ) break;
     console.log(n);
}
```

3) Rehaga los ejercicios de Iterables de Semana 10b pero usando generadores

4) Para la classe `Stream` hecha en clase, escriba un método `first( p )` que encuentra el primer elemento en el `Stream` que cumpla el predicado `p`. Si lo encuentra, lo sigue retornando infinitamente (después vemos cómo "pararlo", pero si un generador evalúa un **`return`** entonces se termina la búsqueda, entonces puede probar esa opción). Si no lo encuentra el `Stream` sigue generando infinitamente.