Gonzalo Canavesio

[20 pt.] ¿Qué características de los lenguajes de scripting se pueden observar en el siguiente programa? Siempre que sea posible, cite los fragmentos específicos de código en los que se observan las propiedades que mencione.

```
tell application "Finder"
    set passAns to "app123"
    set userAns to "John"
    if the text returned of (display dialog "Username" default answer "") is userAns then
        display dialog "Correct" buttons {"Continue"} default button 1
        if the text returned of (display dialog "Username : John" & return & "Password" default answer ""
buttons {"Continue"} default button 1 with hidden answer) is passAns then
            display dialog "Access granted" buttons {"OK"} default button 1
        else
            display dialog "Incorrect password" buttons {"OK"} default button 1
        end if
    else
        display dialog "Incorrect username" buttons {"OK"} default button 1
    end if
end tell
```

- Poco verboso para todo lo que hace
- Construcciones de alto nivel como buttons, dialog, etc
- Sin declaraciones de tipos en las variables
- Fácil interacción con el entorno mediante display (supongo yo)

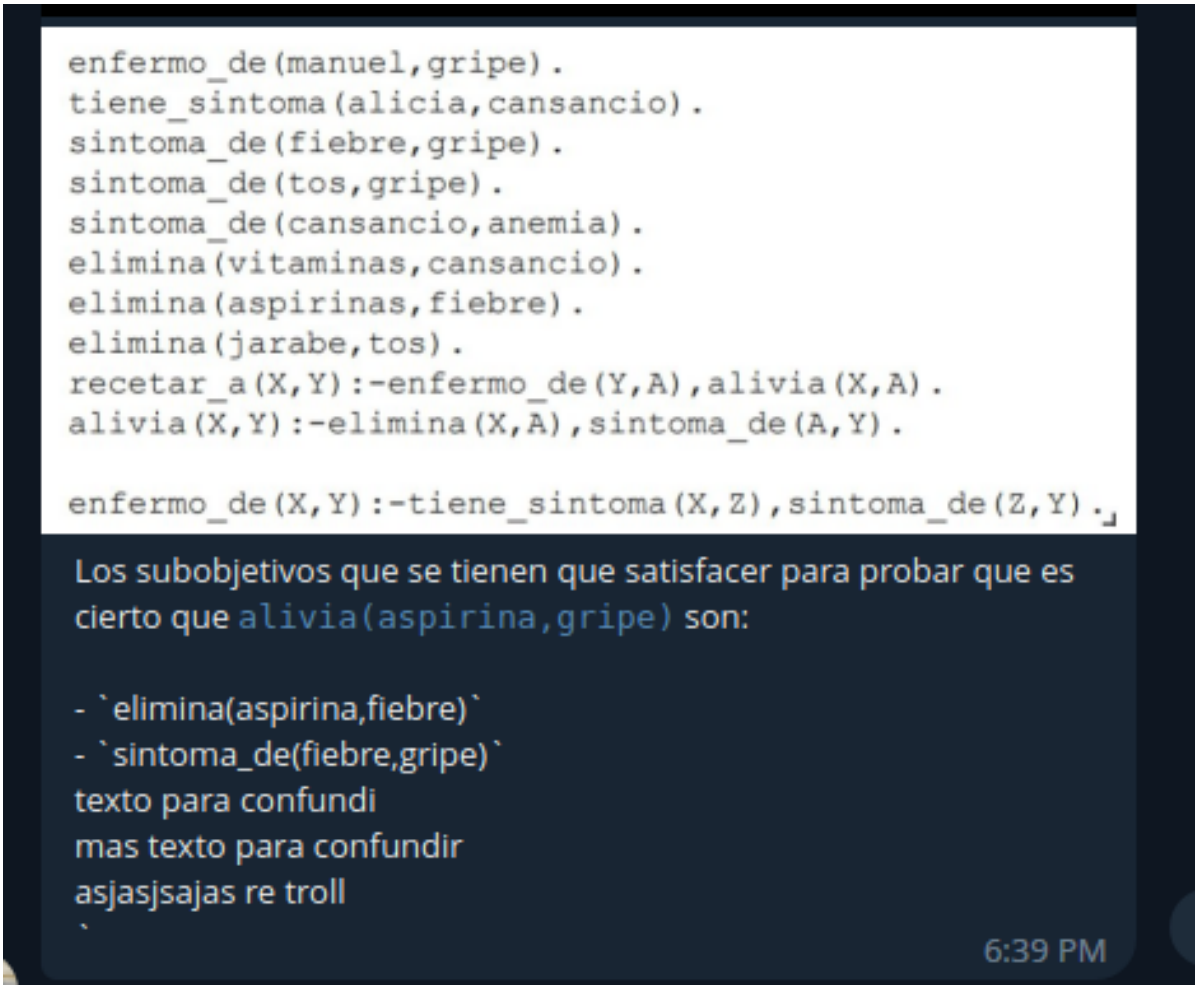
6:39 PM

[20 pt.] El siguiente código forma parte de un código de framework. ¿Qué tipo de elemento vamos a encontrar dentro de la etiqueta Router?

```
import { BrowserRouter as Router } from 'react-router-dom'

const App = () {
  render() {
    return (
        <Router>
           // your routes here
        </Router>
    )
  }
}
```

Dentro de Router se va a encontrar un elemento de tipo hot-spot ya que es algo que cada programador modifica dependiendo de las necesidades de su proyecto

6:39 PM


enfermo_de(manuel,gripe).
tiene_sintoma(alicia,cansancio).
sintoma_de(fiebre,gripe).
sintoma_de(tos,gripe).
sintoma_de(cansancio,anemia).
elimina(vitaminas,cansancio).
elimina(aspirinas,fiebre).
elimina(jarabe,tos).
recetar_a(X,Y):-enfermo_de(Y,A),alivia(X,A).
alivia(X,Y):-elimina(X,A),sintoma_de(A,Y).
enfermo_de(X,Y):-tiene_sintoma(X,Z),sintoma_de(Z,Y).
Los subobjetivos que se tienen que satisfacer para probar que es cierto que alivia(aspirina,gripe) son:
- `elimina(aspirina,fiebre)`
- `sintoma_de(fiebre,gripe)`
texto para confundi
mas texto para confundir
asjasjsajas re troll
`
6:39 PM

[10 pt.] Según el siguiente texto:

> *Hewitt argued against adding the requirement that messages must arrive in the order in which they are sent to the actor. If output message ordering is desired, then it can be modeled by a queue actor that provides this functionality. Such a queue actor would queue the messages that arrived so that they could be retrieved in FIFO order. So if an actor X sent a message M1 to an actor Y, and later X sent another message M2 to Y, there is no requirement that M1 arrives at Y before M2. In this respect the actor model mirrors packet switching systems which do not guarantee that packets must be received in the order sent. Not providing the order of delivery guarantee allows packet switching to buffer packets, use multiple paths to send packets, resend damaged packets, and to provide other optimizations.*

¿con qué tipo de mensajes se comunican los actores?

Los actores se comunican por mensajes asincronos en los cuales no se garantiza su orden, pero si se garantiza que serán almacenados en un buffer hasta ser recibidos por el actor correspondiente

6:39 PM

[30 pt.] Comente el siguiente texto, comparando las estrategias de manejo de excepciones y la filosofía "let it crash". En su justificación, incorpore los principios de diseño de resiliencia y elasticidad propios de la orientación a actores. Mencione cómo se relaciona la programación defensiva con una filosofía "let it crash".

Si lo desea, puede rendir este ejercicio mediante un comentario oral.

> *Erlang is designed with a mechanism that makes it easy for external processes to monitor for crashes (or hardware failures), rather than an in-process mechanism like exception handling used in many other programming languages. Crashes are reported like other messages, which is the only way processes can communicate with each other, and subprocesses can be spawned cheaply. The "let it crash"philosophy prefers that a process be completely restarted rather than trying to recover from a serious failure. Though it still requires handling of errors, this philosophy results in less code devoted to defensive programming where error-handling code is highly contextual and specific.*

La filosofia de let it crash dice que es preferible que un proceso sea reiniciado completamente antes que se intente "recuperar el programa" luego de un fallo muy serio mediante el uso de excepciones.
Esta filosofia busca no promover el uso de programación defensiva ya que complica el seguimiento del flujo del programa, además de que los mecanismos normalmente utilizados para la programación defensiva son muy especificos y dependen del contexto particular, pero la filosofia let it crash es más general y se aplica en cualquier caso de error.
Al permitir reiniciar los procesos luego de un error, y sumado a que crear subprocesos es barato, se puede ver una caracteristica similar a la resiliencia y elasticidad de los actores. La resiliencia de los actores se refiere a que si un actor se cae o tiene un fallo, el resto del programa sigue funcionando correctamente y el supervisor puede darse cuenta del error y revisarlo, posiblemente creando un nuevo actor que se encargue de completar la tarea inconclusa. Esto último es a parte de a lo que nos referimos cuando hablamos de elasticidad en los actores, y es que se puede aumentar y reducir la cantidad de actores según necesitemos sin mucha complicación, cosa que también se puede hacer en Erlang de manera similar creando procesos y reiniciandolos si fuera necesario

6:39 PM

1. **[10 pt.]** ¿Qué características de los lenguajes de scripting se pueden observar en el siguiente programa? Siempre que sea posible, cite los fragmentos específicos de código en los que se observan las propiedades que mencione.

```
set found ''
cat /etc/fstab | while read dev mnt rest
  if test "$mnt" = "/"
    set found $dev
  end
end
```

- Poco verboso
- Buena interacción con otras aplicaciones como cat
- Construcciones de alto nivel como read
- Fácil manejo de strings (Por la comparación de strings)

6:39 PM

2. **[15 pt.]** Lea el siguiente texto:

> *If Await.Result() is called at any point before the Future has completed, the Future becomes blocking. If you instead use onComplete, onSuccess, onFailure, map, or flatMap (and some other methods), you are registering a callback function that will occur when the Future returns. Thus, the Future is non-blocking. Use non-blocking Futures with callbacks whenever possible.*

Este texto nos habla de una propiedad interesante de los futuros. ¿Qué efectos tiene esta propiedad con respecto a la eficiencia de los programas orientados a actores? ¿Con qué decisión de diseño de la orientación a actores la pueden relacionar, y por qué?

Esta propiedad aumenta la eficiencia de los programas de actores debido a que no bloquean la ejecución del programa, cosa que si sucedería si usamos Await.result(), el programa se bloquearía hasta obtener el valor del futuro en cuestión y eso demoraría toda la ejecución del programa. Esta propiedad se puede relacionar con la decision de diseño "responsive", ya que el programa esta disponible cuando es necesario y responde de manera correcta a los cambios, sin presentar ningún bloqueo

6:39 PM

5. **[15 pt.]** En el siguiente texto se explica el comportamiento de los streams en programación reactiva.

> *In computing, reactive programming is a declarative programming paradigm concerned with data streams and the propagation of change. With this paradigm, it's possible to express static (e.g., arrays) or dynamic (e.g., event emitters) data streams with ease, and also communicate that an inferred dependency within the associated execution model exists, which facilitates the automatic propagation of the changed data flow.*
> *For example, in an imperative programming setting,* `a := b + c` *would mean that a is being assigned the result of* `b + c` *in the instant the expression is evaluated, and later, the values of* `b` *and* `c` *can be changed with no effect on the value of* `a`*. On the other hand, in reactive programming, the value of* `a` *is automatically updated whenever the values of* `b` *or* `c` *change, without the program having to explicitly re-execute the statement* `a := b + c` *to determine the presently assigned value of* `a`*.*

Según esta definición, explique qué se imprimirá al ejecutar el siguiente programa si el operador "=" se interpreta como el operador de asignación propio de la programación imperativa, o bien si se interpreta como un operador reactivo, que cambia el valor de la variable del lado izquierdo del operador no solamente cuando se hace la asignación explícitamente, sino también cuando las variables referenciadas en la parte derecha del operador cambian.

```
var b = 1
var c = 2
var a = b + c
b = 10
console.log(a)
```

Programación normal se imprime 3...
Programación reactiva se imprime 12

6:39 PM

6. **[10 pt.]** El siguiente código forma parte de un código de framework. ¿Cómo se llama la parte del código en la que se encuentra el string "`/user/:id`"?

```
<div id="app">
  <router-view></router-view>
</div>
...

<script>
...
const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}

const router = new VueRouter({
  routes: [
    { path: '/user/:id', component: User }
  ]
})
...
</script>
```

Hot spot porque es algo que modifica el programador dependiendo de como quiera desarrollar su aplicación

6:39 PM

7. **[5 pt.]** En el siguiente programa, identifique los mecanismos de programación defensiva, y **[10 pt.]** explique qué tipo de inseguridad se está tratando de cubrir con esos mecanismos. **[5 pt.]** ¿Cómo se escribiría este programa en programación ofensiva? ¿Cuál es el objetivo de seguridad de escribir un programa como este en su versión ofensiva?

```
static void CreateRandomPermutation(int numbers[], int nNumbers)
{
    assert(numbers != NULL);
    assert(nNumbers >= 0);

    for (int i=0; i<nNumbers; i++)
        numbers[i] = i;

    for (int src=1; src<nNumbers; src++)
    {
        const int dst = GetRandomNumberIn(0, src);
        SwapNumbers(numbers, src, dst);
    }
}
```

El uso de guardas/assert es el mecanismo de programación defensiva que se esta usando para evitar una inseguridad de tipos, debido a que numbers debe ser no nulo para poder después acceder a sus elementos, y nNumbers debe ser un número natural según parece, ya que expresa el largo del arreglo. 6:39 PM

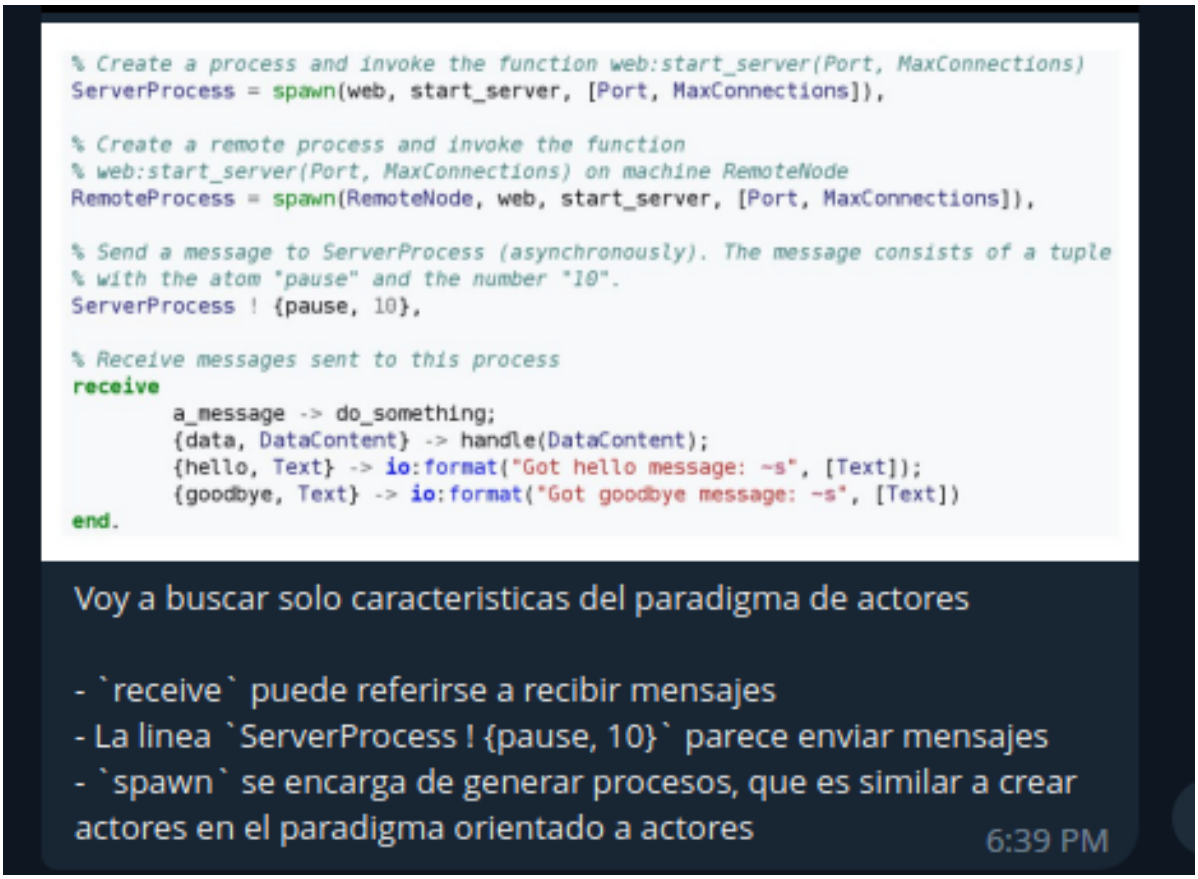
7. **[5 pt.]** En el siguiente programa, identifique los mecanismos de programación defensiva, y **[10 pt.]** explique qué tipo de inseguridad se está tratando de cubrir con esos mecanismos. **[5 pt.]** ¿Cómo se escribiría este programa en programación ofensiva? ¿Cuál es el objetivo de seguridad de escribir un programa como este en su versión ofensiva?

```
static void CreateRandomPermutation(int numbers[], int nNumbers)
{



    for (int i=0; i<nNumbers; i++)
        numbers[i] = i;

    for (int src=1; src<nNumbers; src++)
    {
        const int dst = GetRandomNumberIn(0, src);
        SwapNumbers(numbers, src, dst);
    }
}
```

Version ofensiva del programa

El objetivo de seguridad de escribir el programa en su version ofensiva es que en caso de que surga un error, se debe rastrear ese error para encontrar el origen y analizar cual es la razón por la que sucede para poder actualizar el programa y que sea imposible generar nuevamente ese error

6:39 PM



```
% Create a process and invoke the function web:start_server(Port, MaxConnections)
ServerProcess = spawn(web, start_server, [Port, MaxConnections]),

% Create a remote process and invoke the function
% web:start_server(Port, MaxConnections) on machine RemoteNode
RemoteProcess = spawn(RemoteNode, web, start_server, [Port, MaxConnections]),

% Send a message to ServerProcess (asynchronously). The message consists of a tuple
% with the atom "pause" and the number "10".
ServerProcess ! {pause, 10},

% Receive messages sent to this process
receive
        a_message -> do_something;
        {data, DataContent} -> handle(DataContent);
        {hello, Text} -> io:format("Got hello message: ~s", [Text]);
        {goodbye, Text} -> io:format("Got goodbye message: ~s", [Text])
end.
```

Voy a buscar solo caracteristicas del paradigma de actores

- `receive` puede referirse a recibir mensajes
- La linea `ServerProcess ! {pause, 10}` parece enviar mensajes
- `spawn` se encarga de generar procesos, que es similar a crear actores en el paradigma orientado a actores

6:39 PM

3. **[10 pt.]** Dada la siguiente base de conocimiento, ¿qué va a contestar el intérprete si le preguntamos `digiriendo(rana,mosca).`, y cómo va a llegar a su respuesta?

```
digiriendo(X,Y) :- comio(X,Y).
digiriendo(X,Y) :-
        comio(X,Z),
        digiriendo(Z,Y).

comio(mosquito,sangre(juan)).
comio(rana,mosquito).
comio(gaviota,rana).
```

Va a responder `NO`
El camino que va a hacer va a ser largo y me da paja explicarlo, literalmente va a hacer una busqueda en profundidad y no va a encontrar coincidencia entonces va a responder que no 6:39 PM

5. **[25 pt.]** El siguiente texto explica los conceptos de *hot spot* y *frozen spot* en frameworks. También nos explica cómo funcionan los frameworks con orientación a objetos. Cuando instanciamos una aplicación con un framework basado en objetos, terminamos teniendo un sistema de objetos específico para esa aplicación, basado en el sistema de objetos provisto por el framework. Basándose en los conceptos de *concreto* y *abstracto* y *composición* y *subclases* que usa el texto, explique qué componentes de este sistema de objetos se podrían considerar *frozen spots* y cuáles se podrían considerar *hot spots*.

> *Software frameworks consist of frozen spots and hot spots. Frozen spots define the overall architecture of a software system, that is to say its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. Hot spots represent those parts where the programmers using the framework add their own code to add the functionality specific to their own project.*
>
> *In an object-oriented environment, a framework consists of abstract and concrete classes. Instantiation of such a framework consists of composing and subclassing the existing classes.*

Frozen spots son las clases abstractas y concretas provistas por el framework

Hot spot son clases derivadas de las clases anteriores o instanciaciones de objetos que sean de esas clases

6:39 PM