**FACULTAD DE INGENIERIA DE SISTEMAS**

**ESTUDIO GENERALES**

**TEMA:**

**CLOJURE**

**CURSO DE:**

**ALGORITMICA**

**TRABAJO FINAL**

**PROFESOR:**

**CORONEL CASTILLO ERIC GUSTAVO**

**INTEGRANTES:**

* **PROSCOPIO FERNANDEZ ANTHONY**
* **LOZANO OBLITAS ABRAHAM**
* **CARRION SILVA ALEXIS**
* **ALDUDE MARIANNE**

**LOS OLIVOS – PERÚ**

**2017**

**CLOJURE**

**PRINCIPIOS**

Rich Hickey describe el desarrollo de Clojure como la búsqueda de un lenguaje que no pudo encontrar: un lisp funcional por defecto, integrado sobre un entorno robusto en lugar de ser su propia plataforma, y con la programación concurrente en mente.

Asimismo, en principio se rechaza la orientación a objetos, ofreciendo un enfoque en el que los programas se expresan como la aplicación de funciones sobre datos, más que como la interacción entre entidades mutables que mezclan representación de datos, y comportamiento. Por otra parte, características tales como instanciabilidad, polimorfismo e interfaces son efectivamente parte del lenguaje.

**DESCRIPCION**

Clojure es un dialecto de LISP. Con la manera de que está construido sobre el virtual machine de Java JVM. Es decir que el código se puede compilar atraves de java bytecode que es ejecutado por la VM.

Esto entre otras cosas nos permite tener el código clojure dentro de una aplicación java, y vice-versa. Asi nos permite acceder a clases java desde closure.

**ATOMOS Y LISTAS**

Como LISP es un lenguaje de propósito general y se basa en conceptos muy simples, nos permiten construir sobre este lenguaje aplicaciones de diversos paradigmas.

Si bien se utiliza para trabajar con el paradigma funcional, es un principio, por sus capacidades de meta programación que nos permitiría implementar cualquier paradigma.

Los dos conceptos básicos o tipos de expresiones en clojure son:

• Átomo: Son como los tipos básicos de otros lenguajes. COMO UN NÚMERO, UN STRING, UN CARACTER, ETC.

• Lista: Un conjunto de elementos, que podrían SER ÁTOMOS O LISTAS A SU VEZ.

EJEMPLOS:

Un número:

user=> 42

42

String:

user=> "Hola Clojure!"

"Hola Clojure!"

La expresión de tipo lista se escribe de esta forma:

(1 2 3 4)

**¿Y EL CODIGO?**

¿Pero entonces donde está el comportamiento? Ahí está lo que podemos llamar interesante.

En clojure (al igual que en LISP) el código y los datos se representan de la misma forma,No hay distinción entre ellos.

Por ejemplo, si quisiéramos utilizar una función first (lista) -> elemento es decir que dada una lista retorna el primer elemento, en otros lenguajes haríamos:

First (["a" "b" "c"])

Teniendo una construcción diferente en el lenguaje nombre\_funcion (argumentos)

O en objetos:

["a" "b" "c"].first()

Con la construcción diferente que debe ser variable.mensaje()

En Clojure esta sintaxis especial no existe. Ya que las únicas expresiones son las listas y átomos. Entonces tenemos que escribir la invocación a la función con esas dos herramientas.

Queda algo así...

user=> (first ["a" "b" "c"])

"a"

Es decir que la invocación de la función first con la lista que contiene "a", "b" y "c", se puede pensar a su vez como una lista, donde

• el primer elemento es el nombre de la función

• los siguientes elementos son los argumentos.

Ya que el lenguaje al evaluar la lista, trata de forma especial el primer elemento, e intenta buscar una función para ese nombre, y ejecutarla.

Una especie de dispatching.

**MANIPULANDO CODIGO**

Juguemos un poco con manipular el código ya que se puede tratar como datos:

user=> (rest [1 2 3])

(2 3)

user=> '(rest [1 2 3])

(rest [1 2 3])

user=> (first '(rest [1 2 3]))

rest

rest es una función que nos da una lista donde retorna todos los elementos excepto el primero.

El operador comilla simple ' retorna la lista que precede, sin evaluarla. Es decir, evita el dispatching. Y es lo que nos permite tratar el código como una estructura de datos.

En esto caso, dado una invocación a rest con una lista, estamos obteniendo el nombre de la función, rest. En este caso tenemos nosotros codificamos todas las lineas. Pero piensen que el poder de esto está en que podría llegar como parámetro, y ser cualquier cosa. Como alto orden.

**DEFINIENDO FUNCIONES**

Cómo definimos entonces nuestras propias funciones? Con átomos o listas.

Entonces, nuestra definición de función va a tener que ser expresada como una lista.

Definamos la suma. La función:

• Tiene un nombre: suma

• Recibe dos números como parámetros.

• Se define como a + b

Todo eso lo tenemos que escribir como una lista.

En clojure existe una función especial llamada defn que permite definir nuevas funciones.

Entonces la forma de declarar una nueva función es:

user=> (defn sumar [numero otro] (+ numero otro))

#'user/sumar

user=> (sumar 2 5)

7

Noten la sintaxis de defn es:

(defn nombreFuncion vectorArgumentos Cuerpo)