

**Lenguaje de Programación:**

Clojure

**Integrantes:**

* Agila Pinto Nicole Salome.
* Benalcázar García Sebastián Elías.
* Dávalos Carrera Josué Eulises

**Materia:**

Lenguajes de Programación.

**Profesor:**

Saraguro Bravo Rodrigo Alexander.

**Paralelo:**

Paralelo 02.

**Fecha de entrega:**

Lunes 9 de Noviembre del 2020.

Texto

Descripción generada automáticamente

# **OBJETIVO GENERAL**

Construir un analizador léxico para un lenguaje de programación específico, en este caso Clojure, con la finalidad de ayudar a los nuevos aprendices a identificar las principales funciones de manejo de datos soportados.

# **OBJETIVOS ESPECÍFICOS**

* Aplicar expresiones regulares para reconocer la sintaxis léxica de los componentes de la plataforma.
* Validar la declaración de variables y tipos de datos soportados por Clojure.
* Definir una gramática específica que reconozca las estructuras de control básicas.
* Establecer tokens que permitan reconocer las instrucciones establecidas.
* Validar la declaración de operaciones matemáticas, lógicas y de comparación.

# **ALCANCE**

Para la viabilidad del proyecto, será analizará la declaración de variables y su inicialización, así como los tipos de datos que soporta el lenguaje de programación. También se incluirá la definición de las condiciones bajo las que JavaScript reconoce el enunciado de una operación matemática junto con sus operadores y sus estructuras de control.

## **INICIALIZACIÓN DE VARIABLES**

**(** **def** nombre valor **)**

El nombre de la variable puede contener letras, números y guiones bajos. Deben comenzar o con letras o con guion bajo. Se distingue entre mayúsculas o minúscula.

## **TIPOS DE DATOS**

|  |  |  |
| --- | --- | --- |
| **Datos primitivos** | **Descripción** | **Ejemplo** |
| Entero | Tipo de dato entero, pueden ser short, long e int. | ( def x 42 ) |
| Flotante | Numeros con coma flotante. | ( def y -1.5 ) |
| Booleano | Dato booleano, puede tener el valor de true o false. | ( def status true ) |
| Char | Define un solo carácter literal. | /e |
| Null | Representa valor NULL | nil |

|  |  |  |
| --- | --- | --- |
| **Datos estructurados** | **Descripción** | **Ejemplo** |
| String | Cadena de caracteres | (def str1 “metodo 1”) |
| (def str2 (srt “metodo” “2”)) |
| Lista | Estructura que se usa para guardar datos | (def lista1 ‘(1 2 3 4)) |
| (def lista2 (list (1 2 3 4))) |
| Conjunto | Estructura que almacena valores unicos | (def set1 (set ‘(1 5 9 6))) |
| (der set2 #{1 5 9 6}) |
| Mapa | Coleccion con pares de valores, clave y valor. Hay dos tipos de mapas, hashed y sorted. | (def mapa1 {“a” “a1”, “b” “b1”}) |
| (def mapa2 (hash-map "a" "b")) |
| Vector | Coleccion que almacena valores, se pueden indexar con numeros consecutivos | (def vector1 (vector 1 2 3)) |
| (def vector2 [1 2 3]) |

## **OPERADORES ARITMÉTICOS**

|  |  |
| --- | --- |
| OPERADORES DESCRIPCIÓN | |
| + | Adición |
| - | Sustracción |
| \* | Multiplicación |
| \*\* | Exponenciación |
| / | División |

## **OPERADORES LÓGICOS:**

|  |  |
| --- | --- |
| OPERADORES DESCRIPCIÓN | |
| && | Y lógico |
| || | O lógico |
| ! | No lógico |

## **OPERADORES DE COMPARACIÓN**

|  |  |
| --- | --- |
| OPERADOR DESCRIPCIÓN | |
| == | Igual a |
| === | Valor igual y tipo igual |
| != | No igual |

## **METODOS DE IMPRESIÓN, LECTURA DE DATOS**

|  |  |  |
| --- | --- | --- |
| console.log(str) | Se usa para mostrar mensajes generales de registro de información por consola. | console.log(“Hola mundo”) |

## **ESTRUCTURAS DE CONTROL BASICAS**

|  |  |  |
| --- | --- | --- |
| **FOR** |  |  |
| **WHILE** |  |  |
| **IF-ELSE** |  |  |

1. **MÉTODOS DE DATOS ESTRUCTURADOS**

|  |  |  |  |
| --- | --- | --- | --- |
| List\* | Crea una nueva lista que contiene los elementos antepuestos al resto, el último de los cuales se tratará como una secuencia. En este caso agregará a lista 2, los valores de lista 1, después de los valores de lista 2. | (list\* lista2 [lista1]) | (list\* 1 [2,3])  -> (1 2 3) |
| first | Devuelve el primer valor dentro de una lista. | (first lista) | (first (list 1 2,3))  -> 1 |

## **DEFINICIÓN DE FUNCIONES**

**(** **defn** nombre **[argumentos]**   
(bloque de código1)   
(bloque de código2)

(bloque de códigoN) **)**

Para definir funciones en clojure se necesita seguir la estructura anterior. Como cabecera de la función se inicia con el prefijo **defn** seguido del nombre de la función, mismo que debe cumplir las mismas reglas que los nombres de variables, y finalmente para terminar con la cabecera de la función se debe incluir entre corchetes los argumentos separados por espacio y en caso de no necesitar argumentos enviar los corchetes vacíos.

Por otra parte, para el cuerpo de la función es son necesarios los bloques de código pertinentes encapsulados entre paréntesis línea por línea a la vez. Y al final se concluye la definición de la función con un paréntesis.

**REFERENCIAS**