UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en sistemas de Información
Sintaxis y semántica de los lenguajes – 2023

TURNO: Noche CURSO: K2055

DOCENTE A CARGO: Ing. Roxana Leituz
AYUDANTE:

TRABAJO PRÁCTICO TEÓRICO N°1
“Comparación de lenguajes de programación”

ÁREA TEMÁTICA: Lenguajes de programación

GRUPO N° 30
Rodríguez Lucas Ariel
Golato Barcia Ivan Nahuel
Sayago Pablo
Rabahia Maron Leonel
Schinca Mauro

FECHA DE VENCIMIENTO: 24/08/2023
FECHA DE PRESENTACIÓN: 24/08/2023
FECHA DE DEVOLUCIÓN: __/__/__

CALIFICACIÓN: _____ FIRMA PROFESOR: _____

Índice

Rust

● Historia

Rust es un lenguaje de programación compilado, de propósito general y multiparadigma (funcional, por procedimientos, imperativo y orientado a objetos). Su desarrollo comenzó en 2006 como un proyecto personal de Graydon Hoare, un empleado de Mozilla, y la empresa al ver el potencial de este nuevo lenguaje comenzó a patrocinarlo en 2009, antes de revelarlo al mundo oficialmente en 2010. Finalmente, su primera versión estable 1.0 fue publicada el 15 de mayo del año 2015, desarrollada por Mozilla.

En agosto de 2020, Mozilla despidió 250 empleados para reestructurar internamente la compañía a causa de la pandemia del Covid-19. Entre ellos, el equipo Servo (un motor de renderizado web escrito en Rust). Esto creó preocupación sobre el futuro de Rust, pues varios de estos empleados contribuían activamente al lenguaje.

La siguiente semana, el equipo desarrollador central de Rust confirmó el impacto que habían causado los despidos al lenguaje, tomando la decisión de entregar los dominios y la propiedad completa del mismo a la fundación Rust.

El 8 de febrero de 2021 se anunció la creación formal de la fundación Rust fundada por AWS, Huawei, Google, Microsoft y Mozilla cuyo objetivo es mantener el proyecto y apoyarlo en áreas como las patentes del proyecto o la infraestructura necesaria.

El objetivo al crear Rust fue facilitar la escritura de código, apuntando a ser similar al tiempo requerido en C++ o incluso mejor (lo que se puede ver en su sintaxis que es muy similar), y a su vez eliminar los problemas de memoria.

● Características principales

- ➢ Es un lenguaje compilado
- ➢ Es un lenguaje multiparadigma
- ➢ Sintaxis parecida a c/c++
- ➢ Inferencia de tipos
- ➢ Inmutabilidad por defecto para cualquier variable que declaremos
- ➢ Todo código comienza a ejecutarse por una función main
- ➢ No utiliza un garbage collector
- ➢ Es de código abierto

● Usos principales

- ➢ Aplicaciones en línea de comandos
- ➢ Creación de modulos webassembly(*)
- ➢ Creación de servidores
- ➢ Programación de dispositivos integrados

(*) Un módulo WebAssembly (también conocido como módulo WASM) es un componente de código binario ejecutable que se utiliza en aplicaciones web para mejorar el rendimiento de las aplicaciones en el navegador. WebAssembly es un estándar abierto y un formato binario compacto que permite ejecutar código de alto rendimiento en navegadores web de manera eficiente.

- ## BNF

(inicio del programa)

<program>: := <item>*

<item>:: =

<function> | <struct> | <enum> | <use_declaration> | <impl_block> | <extern_block> | <mod_item> | <attribute>

(una función)

<function> ::= "fn" <identifier> <function_signature> <block>

<function_signature> ::= "(" [<function_params>] ")" ["->" <return_type>] [<where_clause>]

<function_params>::= <identifier> ":" <type> ("," <identifier> ":" <type>)*

<return_type> ::= <type>

(una expression)

<expression> ::=

// algunas reglas para expresiones son por ejemplo <binary_expression> ,<unary_expression>, etc.

(condición)

<if_statement>   ::= "if" <expression> <block> ["else" <block>]

(bucles)

<while_loop>   ::= "while" <expression> <block>

<for_loop>   ::= "for" <identifier> "in" <expression> <block>

(Caso return)

<return_statement>::= "return" [<expression>] ";"

(declaración)

<type>   ::=

// reglas de declaración de tipos, por ejemplo

<primitive_type>, <struct_type>, etc.

<struct>   ::= "struct" <identifier> "{" [<struct_field> ("," <struct_field>)*] "}"

<struct_field>   ::= <identifier> ":" <type>

<enum>   ::= "enum" <identifier> "{" [<enum_variant> ("," <enum_variant>)*] "}"

<enum_variant>   ::= <identifier> ["(" <variant_data> ")"]

<variant_data>   ::=

 // datos asociados con una variable enum, por ejemplo

<field_type> ("," <field_type>)*)

<use_declaration>::= "use" <identifier> ["::" <identifier>]* ";"

<impl_block>   ::= "impl" [<type>] <block>

<extern_block>   ::= "extern" <block>

<mod_item>   ::= "mod" <identifier> <block>

<attribute>   ::= "#" <identifier> ["(" <attribute_args> ")"]

<attribute_args> ::= lista de argumentos, por ejemplo

<expression> ("," <expression>)*

JavaScript

● Historia

JavaScript es un lenguaje de programación interpretado, basado en prototipos, multiparadigma (orientada a objetos, imperativa y declarativa), de un solo hilo y dinámico. Su primera versión se lanzó en septiembre de 1995 con Netscape Navigator 2.0 bajo el nombre "LiveScript". Sin embargo, en diciembre del mismo año, Netscape se asoció con Sun Microsystems (los creadores de Java) y cambió el nombre del lenguaje a "JavaScript".

Debido a la necesidad de establecer un estándar para el lenguaje, Netscape presentó JavaScript a Ecma International en noviembre de 1996. Ecma International formó un comité técnico, TC39, para estandarizar el lenguaje. El estándar resultante se llamó ECMAScript, y la primera edición se publicó en junio de 1997.

Su objetivo y uso más extendido en la actualidad es como un lenguaje de scripting para páginas web, permitiendo que se hagan modificaciones a una página web dinámicamente del lado del cliente sin la necesidad de la interacción con el servidor, lo que fue extremadamente importante cuando el lenguaje se empezó a usar y dio paso a el desarrollo web como lo conocemos hoy en día. Además, JavaScript se puede usar para otros propósitos, como en el backend gracias a Node.JS.

● Características principales

- ➢ Lenguaje de programación interpretado
- ➢ Dinámicamente tipado
- ➢ Multiplataforma
- ➢ Orientado a objetos
- ➢ Manejo del DOM

● Usos principales

- ➢ Crear scripts para páginas web
- ➢ Desarrollo backend (Node.JS)
- ➢ Desarrollo con aplicaciones Adobe Acrobat
- ➢ Estándar de intercambio de datos (JSON)

- BNF

(inicio de programa)

<program> ::= <statement_list>

<statement_list> ::= <statement> | <statement> <statement_list>

<statement> ::=

<variable_declaration> | <function_declaration>| <if_statement>

| <while_loop>| <for_loop>| <expression_statement>

| <return_statement>| <break_statement>

| <continue_statement>| <block>

(declaración)

<variable_declaration> ::= "var" <identifier> ["=" <expression>] ";"

(función)

<function_declaration> ::= "function" <identifier> "(" [<parameter_list>] ")" <block>

<parameter_list> ::= <identifier> | <identifier> "," <parameter_list>

<logical_or_expression> ::=

<logical_and_expression> | <logical_or_expression> "||" <logical_and_expression>

<logical_and_expression> ::=

<equality_expression> | <logical_and_expression> "&&" <equality_expression>

<equality_expression> ::= <relational_expression>

| <equality_expression> "==" <relational_expression>

| <equality_expression> "!=" <relational_expression>

<relational_expression> ::= <additive_expression>

| <relational_expression> "<" <additive_expression>

| <relational_expression> ">" <additive_expression>

| <relational_expression> "<=" <additive_expression>

| <relational_expression> ">=" <additive_expression>

<additive_expression> ::= <multiplicative_expression>

| <additive_expression> "+" <multiplicative_expression>

| <additive_expression> "-" <multiplicative_expression>

<multiplicative_expression> ::= <unary_expression>

| <multiplicative_expression> "*" <unary_expression>

| <multiplicative_expression> "/" <unary_expression>

| <multiplicative_expression> "%" <unary_expression>

<unary_expression> ::= "-" <unary_expression>

| "!" <unary_expression>

| <primary_expression>

<primary_expression> ::= <identifier>

| <literal>

| "(" <expression> ")"

| <function_call>

<function_call> ::= <identifier> "(" [<argument_list>] ")"

<argument_list> ::= <expression>

| <expression> "," <argument_list>

<return_statement> ::= "return" [<expression>] ";"

<break_statement> ::= "break" ";"

<continue_statement> ::= "continue" ";"

<block> ::= "{" [<statement_list>] "}"

(condicional)

< statement_If>::= "if" <identifier> "(" [<parameter_list>] ")" "("<statement_list>")"

(bucles)

< statement_for>::= "for" "(" (<variable_declaration> | <statement> | null) ";"<Expression> ";" (<Expression> | null) ")"<statement>

<Statement_While> ::= "while" "(" <Expression>")"<statement>

<Expression> ::= <AssignmentExpression>

<AssignmentExpression> ::= <LeftHandSideExpression> <AssignmentOperator> <AssignmentExpression>

| <ConditionalExpression>

Ejemplos de Código y comparaciones

- Ejemplo 1:

Calcular los números primos hasta cierto límite.

En Rust:

```rust
use std::time::{Instant};
fn is_prime(num: u64) -> bool {
    if num <= 1 {
        return false;
    }
    for i in 2..((num as f64).sqrt() as u64 + 1) {
        if num % i == 0 {
            return false;
        }
    }
    true
}

fn main() {      You, 8 minutes ago • first commit …
    // Obtén el tiempo de inicio
    let start_time = Instant::now();
    // Algo que queremos medir
    let limit = 100000; // Número límite hasta el cual se generarán los primos
    let mut primes = Vec::new();

    for num in 2..limit {
        if is_prime(num) {
            primes.push(num);
        }
    }
    println!("Tiempo de ejecución: {:.2?}", start_time.elapsed());
    println!("Primos encontrados: {:?}", primes);
}
```

Resultado:

```
warning: crate `exampleNumbers` should have a snake case name
  |
  = help: convert the identifier to snake case: `example_numbers`
  = note: `#[warn(non_snake_case)]` on by default

warning: `exampleNumbers` (bin "exampleNumbers") generated 1 warning
    Finished release [optimized] target(s) in 0.47s
PS C:\Users\PC1\Documents\Facultad\SSL\rust\jsvsRust\exampleNumbers\target\release> ./exampleNumbers.exe
Tiempo de ejecución: 15.48ms
Primos encontrados: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 8
 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277,
379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 5
```

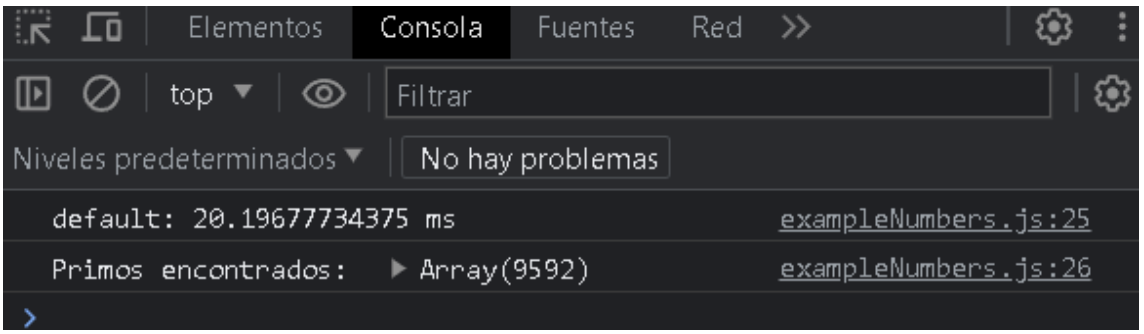
Tiempo de ejecución: 15.48 ms (milisegundos)

En JavaScript:

```javascript
function isPrime(num) {
    if (num <= 1) {
        return false;
    }
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) {
            return false;
        }
    }
    return true;
}

function findPrimes(limit) {
    const primes = [];
    for (let num = 2; num < limit; num++) {
        if (isPrime(num)) {
            primes.push(num);
        }
    }
    return primes;
}
console.time();
const limit = 100000; // Número límite hasta el cual se generarán los primos
const primes = findPrimes(limit);
console.timeEnd();
console.log("Primos encontrados: ", primes);
```

Resultado:



Tiempo de ejecución: 20.197 ms (milisegundos)

- Ejemplo 2:

Calcular la suma de los primeros N números naturales utilizando recursión.

En Rust:

```rust
use std::time::{Instant};
fn sum_natural_numbers(n: u64) -> u64 {
    if n == 0 {
        return 0;
    }
    n + sum_natural_numbers(n - 1)
}

fn main() {
    let n = 10000;
    let result = sum_natural_numbers(n);
    // Obtén el tiempo de inicio
    let start_time = Instant::now();

    println!("La suma de los {} primeros números naturales es: {}", n, result);
    // Obtén el tiempo de finalización
    println!("Tiempo de ejecución: {:.2?}", start_time.elapsed());
}       You, 16 minutes ago • first commit
```

Resultado:

```
PS C:\Users\PC1\Documents\Facultad\SSL\rust\jsvsRust\anotherExampleRust\anotherExample> Cargo build --release
warning: crate `anotherExample` should have a snake case name
  |
  = help: convert the identifier to snake case: `another_example`
  = note: `#[warn(non_snake_case)]` on by default

warning: `anotherExample` (bin "anotherExample") generated 1 warning
    Finished release [optimized] target(s) in 0.01s
PS C:\Users\PC1\Documents\Facultad\SSL\rust\jsvsRust\anotherExampleRust\anotherExample> ./target/release/anotherexample.exe
La suma de los 10000 primeros números naturales es: 50005000
Tiempo de ejecución: 224.40µs
```

Tiempo de ejecución: 224.40 µs (microsegundos)
Que equivalen a 2.244 ms (milisegundos)

En JavaScript:

```
1   console.time();
2   function sumNaturalNumbers(n) {
3       if (n === 0) {
4           return 0;
5       }
6       return n + sumNaturalNumbers(n - 1);
7   }
8
9   const n = 10000;
10  const result = sumNaturalNumbers(n);
11  console.log(`La suma de los ${n} primeros números naturales es: ${result}`);
12  console.log(console.timeEnd());
```

Resultado:

```
La suma de los 10000 primeros números naturales es: 50005000    anotherExamplejs.js:11
default: 3.58203125 ms                                          anotherExamplejs.js:12
undefined                                                       anotherExamplejs.js:12
>
```

Tiempo de ejecución: 3.582 ms (milisegundos)

Conclusión

Rust suele ser mucho más rápido que Javascript a la hora de ejecutar tareas ya que Rust es un lenguaje compilado, por lo que este puede hacer muchas más optimizaciones al código de manera interna, comparado con Javascript que es interpretado.

Por ejemplo, en cuanto a las optimizaciones hechas por cada lenguaje, Rust tiene un poderoso sistema de macros que permite la abstracción sintáctica y la metaprogramación. Las macros pueden generar código en tiempo de compilación basado en patrones o reglas. JavaScript tiene un sistema de macros limitado que se basa en herramientas externas como Babel o TypeScript para transformar el código en el momento de la compilación en función de plug-in o anotaciones.

Además, Rust tiene la ventaja de no utilizar un garbage collector, que es un proceso usado para el manejo de la memoria, lo que hace que tenga un proceso menos consumiendo recursos en su ejecución a diferencia de Javascript que si tiene un garbage collector.