**Sección: CC61**

***| Ciencias de la computación |***

**CC218-TEORÍA DE COMPILADORES**

**Integrantes**

Andrea Abigail Gonzales Astoray | UF20211C56

Nathaly Eliane Anaya Vadillo | U202210644

Gleider Venancio Castro Ataucusi | U20201B122

Diego Tomas Villafuerte Ramirez | U202010546

**2024-02**

**TRABAJO PARCIAL**

**Docente**

Jorge Eduardo Diaz Suarez



CONTENIDO

[**PROBLEMÁTICA** 3](#_Toc178269926)

[1.1 DESCRIPCIÓN DEL PROBLEMA 3](#_Toc178269927)

[1.2 CONSECUENCIAS 3](#_Toc178269928)

[**MOTIVACIÓN** 5](#_Toc178269929)

[**OBJETIVOS** 5](#_Toc178269930)

[**DISEÑO DE LA SINTAXIS Y SEMÁNTICA DEL NUEVO LENGUAJE** 6](#_Toc178269931)

[4.1 DEFINIR LOS ELEMENTOS 6](#_Toc178269932)

[4.2 SIMPLIFICACIÓN DE LA SINTAXIS 7](#_Toc178269933)

[4.3 DISEÑO DEL LENGUAJE AMIGABLE 9](#_Toc178269934)

[4.4 DEFINICIÓN DE REGLAS DE GRAMÁTICA EN ANTLR4 9](#_Toc178269935)

[**REFERENCIAS** 11](#_Toc178269936)

# **PROBLEMÁTICA**

Simplificación de la sintaxis y usabilidad en lenguajes de programación

## DESCRIPCIÓN DEL PROBLEMA

En el contexto del desarrollo de software, la complejidad sintáctica de los lenguajes de programación y la falta de accesibilidad para principiantes son problemas críticos que afectan a una amplia gama de usuarios, incluidos estudiantes, científicos, ingenieros, y profesionales de otras disciplinas. Lenguajes como Python, Go y otros, aunque reconocidos por su potencia y versatilidad, presentan una sintaxis que puede ser difícil de aprender y utilizar, especialmente para aquellos sin formación previa en informática.

Los usuarios nuevos enfrentan desafíos al intentar comprender conceptos básicos de programación, como estructuras de control, gestión de errores y tipos de datos, lo que convierte tareas simples en procesos complicados y desalentadores (Xu et al., 2024). Estos obstáculos son especialmente pronunciados para quienes están acostumbrados a herramientas más intuitivas como Excel, donde las acciones se realizan mediante comandos simples y lógicos. La transición a un lenguaje de programación estructurado suele ser abrupta y frustrante, aumentando la resistencia a aprender y adoptar nuevas habilidades tecnológicas. (Qasse et al., 2021).

## CONSECUENCIAS

* **Baja Adopción y Desmotivación:**

La complejidad de la sintaxis en los lenguajes de programación resulta en una baja tasa de adopción entre nuevos usuarios. Estudiantes y profesionales que se enfrentan a un alto número de errores sintácticos y dificultades técnicas suelen experimentar frustración, lo que reduce su motivación para continuar aprendiendo. Richard-Foy (2023) observa que "la complejidad sintáctica amplifica la frecuencia de errores, lo que puede desmotivar a los principiantes y generar un rechazo hacia la programación".

* **Curva de Aprendizaje Empinada y Pérdida de Productividad:**

Los principiantes, especialmente aquellos en niveles educativos básicos, encuentran que aprender a programar consume más tiempo del previsto debido a la necesidad de dominar no solo la lógica del problema, sino también la estricta sintaxis del lenguaje. Esto no solo ralentiza su aprendizaje, sino que también afecta la productividad general, ya que tareas que podrían automatizarse rápidamente se convierten en procesos tediosos (Xu et al., 2024).

* **Dificultades en la Resolución de Problemas Específicos:**

Los lenguajes de programación tradicionales no siempre ofrecen un enfoque natural para la resolución de problemas específicos, lo cual es un desafío particular para profesionales no informáticos. Por ejemplo, científicos e ingenieros que necesitan manipular datos rápidamente encuentran que los lenguajes tradicionales no se adaptan fácilmente a su lógica de trabajo, incrementando la curva de aprendizaje y el tiempo necesario para aplicar correctamente estas herramientas (Qasse et al., 2021).

* **Falta de Puentes entre Herramientas Conocidas y Programación:**

La falta de integración entre herramientas ampliamente conocidas, como Excel, y los lenguajes de programación, crea un vacío significativo en la experiencia de usuario. Los comandos intuitivos y familiares de Excel no se traducen directamente a los lenguajes de programación, lo que aumenta la complejidad y la necesidad de aprender nuevos paradigmas desde cero (Xu et al., 2024). Esta desconexión refuerza la resistencia al aprendizaje y limita el potencial de los usuarios para automatizar tareas y optimizar sus flujos de trabajo.

# **MOTIVACIÓN**

La motivación para simplificar la sintaxis y mejorar la usabilidad de los lenguajes de programación surge de la necesidad de hacer la programación accesible para principiantes y profesionales de disciplinas no informáticas, como científicos, ingenieros y estudiantes. A menudo, estos usuarios enfrentan grandes barreras al aprender lenguajes tradicionales debido a su complejidad y estructura rígida, lo que resulta en desmotivación y baja adopción. Facilitar esta transición mediante un enfoque que combine comandos familiares, como los de Excel, con elementos básicos de lenguajes como Python ,Go y C++, busca reducir la curva de aprendizaje y empoderar a los usuarios para que apliquen la programación de manera efectiva en sus actividades cotidianas, democratizando así el acceso a estas valiosas habilidades tecnológicas.

# **OBJETIVOS**

* Simplificar la Sintaxis para Principiantes
* Facilitar la Programación para Profesionales No Informáticos
* Reducir la Curva de Aprendizaje
* Promover la Programación en la Educación
* Fomentar una Comunidad de Apoyo

# **DISEÑO DE LA SINTAXIS Y SEMÁNTICA DEL NUEVO LENGUAJE**

## DEFINIR LOS ELEMENTOS

Para el diseño de la sintaxis y semántica del nuevo lenguaje, es fundamental identificar los elementos clave de los lenguajes de programación C++, Go, Excel y Python que se integrarán. A continuación, se presenta una tabla que detalla los elementos comunes y diferenciadores de estos lenguajes, resaltando cómo se pueden combinar para crear un lenguaje accesible y útil para principiantes y profesionales no informáticos.

|  |  |  |
| --- | --- | --- |
| Aspecto | Común entre los Lenguajes | Diferenciador por Lenguaje |
| Sintaxis | - Uso de estructuras de control (if, for, while)  - Definición de funciones y variables | * C++: Sintaxis compleja y detallada  Golang: Limpia y moderna con manejo explícito de errores  Python: Legible y concisa |
| Manejo de Datos | - Soporte para listas y arrays  - Operaciones matemáticas y lógicas | * Excel: Manejo de datos tabulares  Python: Flexibilidad en estructuras de datos como listas y diccionarios |
| Funcionalidad | - Funciones personalizadas  - Operadores matemáticos y lógicos | * Excel: Fórmulas predefinidas * Python: Librerías extensas para cálculos avanzados * Golang: Optimización para concurrencia |
| Manejo de Errores | - Control básico de errores (try/catch o similar) | * C++: Complejo y manual * Golang: Manejo explícito y controlado * Python: Simplificado con try/except |
| Tipos de Datos | - Soporte para enteros, cadenas, booleanos y flotantes | * C++: Tipado estático y fuerte * Python: Tipado dinámico y flexible * Excel: Tipos implícitos en celdas |
| Visualización | - Salida de datos (print, mostrar) | * Excel: Visualización directa en tablas * Python: Gráficas y plots fáciles con librerías * C++/Golang: Menos orientados a visualización |
| Concurrencia | - Uso básico de tareas y procesos | * Golang: Potente en concurrencia y paralelismo * C++: Threads manuales * Python: Asincronía simplificada |
| Fórmulas y Cálculos | - Realización de cálculos matemáticos básicos | * Excel: Uso de fórmulas directas y referencias * Python: Integración de cálculos con código limpio y simple |
| Facilidad de Uso | - Básico acceso a funciones y operaciones comunes | * Python: Más fácil de aprender y leer * Excel: Familiaridad con fórmulas, bajo barrera de entrada |
| Bibliotecas y Paquetes | - Soporte para módulos o paquetes adicionales para extender funcionalidades | * Python: Amplias librerías para todo tipo de tareas * Golang: Librerías optimizadas * C++: Complejidad en integración de bibliotecas |

## SIMPLIFICACIÓN DE LA SINTAXIS

La simplificación de la sintaxis del nuevo lenguaje es crucial para hacerlo accesible a usuarios sin experiencia previa en programación. El objetivo es eliminar las complejidades que presentan los lenguajes tradicionales y proporcionar un entorno de programación intuitivo que permita a los principiantes concentrarse en la resolución de problemas sin las distracciones de una sintaxis estricta.

* **Palabras Clave Intuitivas**: Utilizar palabras en lenguaje natural para comandos comunes, como **mostrar** en lugar de **print**, y **calcular** en lugar de **eval**.

Ejemplo:

Texto

Descripción generada automáticamente

* **Sintaxis Minimalista**: Reducir el uso de caracteres especiales como llaves {}, punto y coma ;, y paréntesis (), que a menudo son una fuente de errores para principiantes.

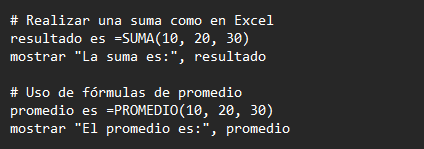
Ejemplo:

Texto

Descripción generada automáticamente

* **Fórmulas Estilo Excel**: Integrar la capacidad de utilizar fórmulas directas como en Excel para cálculos y manipulación de datos, haciendo que el lenguaje se sienta más familiar.

Ejemplo:



* **Tipado Dinámico**: Al igual que en Python, permitir la creación de variables sin la necesidad de declarar tipos, lo que facilita el uso del lenguaje para tareas simples.

Ejemplo:

Texto

Descripción generada automáticamente

* **Manejo Automático de Errores**: Incorporar un manejo de errores simplificado que ofrezca mensajes claros y orientaciones automáticas para corregir errores comunes sin necesidad de una gestión manual complicada.

Ejemplo:

Texto

Descripción generada automáticamente

* **Funciones Predefinidas**: Proveer un conjunto de funciones predefinidas y accesibles que cubran las necesidades más frecuentes, como cálculos, visualización de datos y control de flujo, sin requerir configuraciones adicionales.

Ejemplo:

Texto

Descripción generada automáticamente

## DISEÑO DEL LENGUAJE AMIGABLE

El diseño del lenguaje amigable se centra en crear un entorno de programación intuitivo, accesible y atractivo para los usuarios sin experiencia previa en programación. Este diseño tiene como objetivo reducir la barrera de entrada y facilitar el aprendizaje mediante la incorporación de elementos familiares y simplificados. A continuación, se detallan los componentes clave del diseño del lenguaje amigable:

|  |  |  |
| --- | --- | --- |
| Componente | Descripción | Ejemplo |
| Interfaz Visual | Retroalimentación inmediata con resultados visibles en tiempo real. | Área de trabajo que muestra resultados al instante. |
| Comandos Simples | Uso de lenguaje natural para escribir instrucciones. | mostrar "Hola"  calcular promedio (10, 20, 30) |
| Plantillas y Autocompletado | Plantillas de código y sugerencias automáticas. | Sugerencia de suma, resta al escribir calcular. |
| Biblioteca de Recursos Visuales | Gráficos, tablas y diagramas para visualizar datos fácilmente. | Comando graficar datos para crear gráficos interactivos. |

## DEFINICIÓN DE REGLAS DE GRAMÁTICA EN ANTLR4

Para diseñar la sintaxis y semántica del nuevo lenguaje, ANTLR4 (Another Tool for Language Recognition) se utiliza para definir las reglas de gramática que dictan cómo se estructuran los comandos y las funciones. Estas reglas permiten crear un analizador sintáctico que puede interpretar y ejecutar el código escrito en el nuevo lenguaje, transformando comandos simplificados en operaciones efectivas.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Link Colab: <https://colab.research.google.com/drive/1FUXH72XqofANqh9Bu0jzv0KHA86U7HUQ?usp=sharing>

**Reglas clave:**

1. Estructura General: El programa está compuesto por expresiones (expr), que pueden ser operaciones, funciones, asignaciones, condicionales, y bucles.
2. Funciones: Usamos la palabra clave como DEFINIR para declarar funciones con parámetros y bloques de código.

****

1. Operaciones: Las expresiones incluyen operaciones aritméticas, asignaciones, condicionales, y bucles como por ejemplo: MIENTRAS y PARA.

**Texto

Descripción generada automáticamente**

**Ejemplo de código en el lenguaje:**

Texto

Descripción generada automáticamente

**Árbol Sintáctico del Ejemplo:**

A continuación, mostramos el árbol sintáctico generado por ANTLR4 a partir del ejemplo anterior:

Este árbol muestra la estructura jerárquica del código, donde la función suma contiene expresiones como MOSTRAR y la llamada a MAXIMO.

Diagrama

Descripción generada automáticamente

# **REFERENCIAS**

Xu, S., Li, Z., Mei, K., & Zhang, Y. (2024). *AIOS Compiler: LLM as Interpreter for Natural Language Programming and Flow Programming of AI Agents*. arXiv. <https://doi.org/10.48550/arXiv.2405.06907>

Qasse, I., Mishra, S., & Hamdaqa, M. (2021). *Chat2Code: Towards conversational concrete syntax for model specification and code generation, the case of smart contracts*. arXiv. <https://doi.org/10.48550/arXiv.2112.11101>

Richard-Foy, J. (2023). *State of structural typing support in Scala 3.3.0*. arXiv. <https://doi.org/10.48550/arXiv.2311.11105>