UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

CC218-TEORÍA DE COMPILADORES

| Ciencias de la computación |

Sección: CC97

TRABAJO PARCIAL

Docente

Jorge Eduardo Diaz Suarez

Integrantes

Andrea Abigail Gonzales Astoray | UF20211C56

Nathaly Eliane Anaya Vadillo | U202210644

Gleider Venancio Castro Ataucusi | U20201B122

Diego Tomas Villafuerte Ramirez | U202010546

2024-02

CONTENIDO

1. Problemática 3
1.1. Descripción del problema 3
1.2. Consecuencias 4
2. Motivación 5
3. Objetivos 5
4. Diseño de la sintaxis y semántica del nuevo lenguaje 6
4.1. Definir los elementos 6
4.2. Simplificación de la sintaxis 7
4.3. Diseño del lenguaje amigable 8
4.4. Definición de reglas de gramática en antlr4 9

1. Problemática

Simplificación de la sintaxis y usabilidad en lenguajes de programación

1.1. Descripción del problema

En el desarrollo de software, la complejidad sintáctica sigue siendo un obstáculo significativo para los nuevos programadores. Aunque lenguajes como Python y Go son elogiados por su versatilidad y poder, su sintaxis puede presentar barreras para los estudiantes y profesionales sin experiencia previa en informática. Esto incluye la dificultad para comprender estructuras de control, manejo de errores y tipos de datos, que son conceptos clave en cualquier lenguaje de programación estructurado. Las personas que provienen de disciplinas como la biología, las ciencias o la ingeniería, que utilizan herramientas como Excel o MATLAB, suelen encontrar la transición a un lenguaje de programación estructurado-abrumadora y frustrante (Ade-Ibijola, 2019).

Los usuarios novatos, al aprender lenguajes como Python, suelen cometer errores comunes, como fallos en la estructura lógica, errores de sintaxis y dificultades para implementar estructuras básicas como bucles y condicionales. Estos errores a menudo desalientan a los principiantes, ralentizando su progreso y generando frustración. Un estudio realizado por (Jegede et al., 2023) reveló que los estudiantes enfrentan grandes desafíos al intentar integrar conceptos de control de flujo en sus programas, lo que convierte tareas simples en procesos complicados.

Además, en disciplinas como la bioinformática, donde Python ha sido adoptado ampliamente, la complejidad y diversidad de las bibliotecas disponibles pueden abrumar a los nuevos usuarios. La curva de aprendizaje es empinada porque, aunque Python es conocido por su simplicidad relativa, las funciones avanzadas y la amplia gama de bibliotecas necesarias para la investigación pueden dificultar el dominio del lenguaje (Ekmekci et al., 2016). Esto provoca una resistencia significativa a adoptar nuevas habilidades tecnológicas en áreas no informáticas.

Este problema se agrava cuando los nuevos usuarios no solo deben aprender la lógica del lenguaje, sino también lidiar con una sintaxis estricta, lo que los obliga a corregir numerosos errores menores antes de poder completar una tarea básica. Por ejemplo, Alqadi y Maletic (2021) señalan que, aunque Python facilita la resolución de problemas complejos, la falta de comprensión de las estructuras sintácticas clave puede llevar a un

estancamiento en el aprendizaje, lo que incrementa la tasa de deserción en cursos de introducción a la programación (Ade-Ibijola, 2019).

1.2. 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 resolver problemas específicos, 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.

2. Motivación

La motivación detrás de simplificar la sintaxis y mejorar la usabilidad de los lenguajes de programación radica en la creciente necesidad de hacer que la programación sea accesible a un público más amplio, que incluye tanto a principiantes como a profesionales en campos no directamente relacionados con la informática, como científicos, ingenieros y estudiantes. Estos usuarios suelen enfrentarse a grandes dificultades cuando intentan aprender lenguajes de programación tradicionales, debido a la complejidad de su sintaxis y a la rigidez de su estructura. Estas barreras provocan frustración, desmotivación e incluso el abandono del aprendizaje, limitando el acceso a habilidades tecnológicas que podrían ser extremadamente valiosas en sus respectivos campos. Por ello, la creación de un enfoque que mezcle comandos familiares, como aquellos presentes en herramientas ampliamente utilizadas como Excel, con elementos esenciales de lenguajes accesibles como Python y Go, tiene como objetivo reducir la curva de aprendizaje. Este enfoque facilitaría una transición más fluida hacia el mundo de la programación, empoderando a los usuarios para aplicar estas habilidades en sus actividades cotidianas de manera eficiente y efectiva. Al democratizar el acceso a la programación, no solo se favorece una mayor adopción de estas herramientas, sino que también se promueve la innovación y la resolución de problemas en una variedad de disciplinas, transformando la programación de una barrera a una herramienta accesible para todos.

3. 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

4. Diseño de la sintaxis y semántica del nuevo lenguaje

4.1. 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**

4.2. 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:

```
# Mostrar un mensaje en pantalla
mostrar "Hola, mundo!"

# Calcular una operación simple
resultado es calcular suma(5, 3)
mostrar "El resultado de la suma es:", resultado
```

- **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:

```
# Definir una función sin paréntesis ni llaves
definir saludo es:
    mostrar "¡Bienvenido al nuevo lenguaje!"

# Llamar a la función sin necesidad de paréntesis
saludo
```

- **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:

```
# Realizar una suma como en Excel
resultado es =SUMA(10, 20, 30)
mostrar "La suma es:", resultado

# Uso de fórmulas de promedio
promedio es =PROMEDIO(10, 20, 30)
mostrar "El promedio es:", promedio
```

- **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:

```
# Crear variables sin especificar tipo
numero es 100
texto es "Este es un texto"

mostrar "Número:", numero
mostrar "Texto:", texto
```

- **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:

```
# Intentar dividir por cero y manejarlo automáticamente
resultado es calcular division(10, 0)
mostrar "El resultado de la división es:", resultado
# Mensaje automático: "Error: División por cero. Por favor, verifique los valores."
```

- **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:

```
# Usar funciones predefinidas para calcular el máximo y mínimo
maximo es calcular maximo(5, 10, 20)
minimo es calcular minimo(5, 10, 20)

mostrar "El valor máximo es:", maximo
mostrar "El valor mínimo es:", minimo
```

4.3. 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.

4.4. 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.

5. REFERENCIAS BIBLIOGRÁFICAS

Ade-Ibijola, A. (2019). Syntactic Generation of Practice Novice Programs in Python. *Communications in Computer and Information Science*, *963*, 158–172. https://doi.org/10.1007/978-3-030-05813-5_11

Ekmekci, B., McAnany, C. E., & Mura, C. (2016). An Introduction to Programming for Bioscientists: A Python-Based Primer. *PLOS Computational Biology*, *12*(6), e1004867. https://doi.org/10.1371/JOURNAL.PCBI.1004867

Jegede, P. O., Olajubu, E. A., Bakare, O. O., Elesemoyo, I. O., & Owolabi, J. (2023). Analysis of Syntactic Errors of Novice Python Programmers in a Nigeria University. *Lecture Notes in Networks and Systems*, *739 LNNS*, 285–295. https://doi.org/10.1007/978-3-031-37963-5_20