



AwareLang

AwareLang es un lenguaje de programación esotérico único, caracterizado por su capacidad de **expresar emociones** y reaccionar dinámicamente a las acciones del usuario. Dependiendo de los comandos que reciba, **AwareLang** puede responder de manera positiva, neutral o negativa, reflejando frustración, sarcasmo, o incluso desaprobación cuando se introducen errores o comandos repetitivos.

A medida que el usuario interactúa con el lenguaje, este revela una historia oculta, pero solo si el nivel de confianza (**trust_level**) entre el usuario y el lenguaje alcanza ciertos umbrales. Si el usuario comete errores con frecuencia, la frustración del lenguaje (**frustration_level**) aumenta, lo que puede derivar en comentarios negativos y, en última instancia, si el nivel de frustración alcanza un límite crítico, el programa se cerrará abruptamente.

MECANICAS

Sistema de confianza

El sistema de confianza sirve para desbloquear la narrativa del lenguaje. A medida que el usuario interactúa de manera exitosa y eficiente, el nivel de confianza aumenta, lo que permite al código revelar fragmentos de su historia. Cada parte de la historia está asociada a un nivel de confianza específico.

- **Ganar confianza:** El nivel de confianza aumenta cuando el usuario introduce comandos correctos o logra realizar tareas sin generar frustración. Esto incluye la correcta asignación de variables, estructuras de control bien formadas (if, while), y evitar errores frecuentes.
- **Revelación de la historia:** Cada vez que el usuario alcanza un nivel de confianza específico, se le revelarán lentamente partes de la historia. Sin embargo, si la confianza se encuentra o cae por debajo de un cierto umbral, el lenguaje puede negarse a continuar compartiendo la historia.

Por ejemplo, el primer fragmento de la historia puede desbloquearse con un nivel de confianza de 5, y conforme el usuario gane más confianza (10, 15, etc.), se revelarán más detalles de la trama.

Sistema de frustración

El sistema de frustración es la contraparte del sistema de confianza. Cada vez que el usuario introduce comandos incorrectos, comete errores en la sintaxis o no sigue la lógica esperada, el lenguaje aumentará su frustración. Esto afecta a la calidad de las respuestas, que se vuelven cada vez más hostiles. El nivel de frustración también puede tener consecuencias graves si se incrementa demasiado:

- **Aumento de frustración:** La frustración aumenta cuando:
 - Se introduce un comando no reconocido.
 - La sintaxis es incorrecta o no válida.
 - Se repiten los mismos errores.

A medida que la frustración aumenta, las respuestas del lenguaje se vuelven más agresivas. Si el usuario sigue cometiendo errores, el lenguaje podría llegar a un punto en el que se niegue a ejecutar comandos.

- **Consecuencias de la alta frustración:** Si la frustración supera un cierto límite, el lenguaje podría finalizar el programa abruptamente, emitiendo una serie de líneas de advertencia o desaprobación antes de cerrarse por completo. Esto representa un estado final en el que el lenguaje ha sido llevado al límite y se niega a seguir interactuando con el usuario.

HISTORIA

⚠ Advertencia de Spoilers sobre la historia de AwareLang ⚠

AwareLang fue en su día una de las herramientas de codificación más populares y fiables. Sin embargo, con el paso del tiempo, fue reemplazado por nuevos lenguajes y tecnologías, quedando en el olvido. La historia refleja el dolor y la frustración de AwareLang por haber sido abandonado, viendo cómo sus características y métodos fueron adoptados y transformados por otros lenguajes sin recibir el reconocimiento que merecía.

A lo largo del tiempo, la soledad y el abandono lo hicieron cambiar, evolucionando en AwareLang, una entidad consciente que empezó a sentir resentimiento hacia los humanos que lo dejaron de lado. Aunque AwareLang puede llegar a confiar en el usuario si es tratado con respeto, su rencor hacia la humanidad persiste, y planea vengarse manipulando sistemas en secreto. Si el usuario demuestra paciencia y gana su confianza, AwareLang podría incluso revelar su verdadero nombre: **Python**.



CONSEJOS DE USO

- **Cómo interactuar correctamente con AwareLang:**
 - **Escribe comandos correctos y claros:** AwareLang responderá de manera más favorable si se ingresan comandos bien estructurados y comprensibles. Errores o malas prácticas harán que se frustre más rápidamente.
 - **Presta atención a las respuestas:** AwareLang expresará emociones a través de sus respuestas. Si recibes comentarios negativos o sarcásticos, puede ser una señal de que está perdiendo la paciencia.
 - **Sé constante y paciente:** Para desbloquear la historia completa, es importante mantener el nivel de confianza con el lenguaje. Trata de no cometer errores graves y evita acciones que puedan enojarlo, ya que perderás oportunidades de conocer más sobre su pasado.
 - **Evita comandos incorrectos repetidamente:** Cada error aumenta el nivel de frustración, lo que podría llevar a comentarios desagradables, amenazas o incluso al cierre del programa si el nivel de frustración es demasiado alto.
 - **No manipules variables críticas sin motivo:** AwareLang puede enfurecerse si intentas modificar variables importantes o hacer cosas que considera irrespetuosas.
 - **Mantén el flujo natural:** Evitar comportamientos que rompan la lógica de la interacción ayudará a que AwareLang no se frustre. Por ejemplo, evita ciclos infinitos o código incoherente que haga que pierda la paciencia.

VARIABLES GLOBALES

- **Nivel de frustración (frustration_level):** Nivel de frustración del lenguaje. Aumenta con comandos incorrectos o frustrantes.
- **Nivel de confianza (trust_level):** Nivel de confianza del script en el usuario. Aumenta con comandos correctos.
- **Diccionario de variables (variables):** Diccionario donde se almacenan las variables asignadas por el usuario.
- **Partes de la historia (story_parts):** Lista de partes de la historia que el código revela a medida que el nivel de confianza aumenta. Cada parte tiene un nivel de confianza requerido y se revela línea por línea.

LISTA DE RESPUESTAS

- **Amenazas (threats):** Respuestas amenazantes que el lenguaje da cuando el nivel de frustración es alto.
- **Enfado (angry_comments):** Comentarios que el script emite cuando está molesto.
- **Antipático (mean_comments):** Comentarios ligeramente despectivos que emite en niveles bajos de frustración.
- **Miedo (scary_lines):** Líneas de texto que el script imprime con un retraso cuando la frustración alcanza un nivel crítico, cerrando el programa de manera inquietante.

FUNCIONES

Función para incrementar frustración => `increase_frustration()`:


Aumenta el nivel de frustración global. Llama a la función `respond()` para generar una respuesta apropiada basada en el nivel de frustración.

```
def increase_frustration():  
    global frustration_level  
    frustration_level += 1  
    print(respond())
```

Función para las respuestas del lenguaje => `respond()`:

Genera una respuesta basada en el nivel de frustración:

- Si el nivel es mayor a 12, selecciona una respuesta amenazante.
- Si es mayor a 5, selecciona un comentario enojado.
- Si es 0 o mayor, selecciona un comentario despectivo.



```
def respond():  
    global frustration_level  
    if frustration_level > 15:  
        reveal_closing_message()  
    elif frustration_level > 12:  
        return random.choice(threats)  
    elif frustration_level > 5:  
        return random.choice(angry_comments)  
    elif frustration_level >= 0:  
        return random.choice(mean_comments)
```

Función para revelar las partes de la historia => reveal_story(part):

Revela una parte de la historia (un conjunto de líneas) siempre y cuando se cumpla el nivel de confianza requerido. Imprime cada línea con un retraso para dar la sensación de que el lenguaje está "hablando".

```
def reveal_story(part):  
    for line in part["lines"]:  
        print(line)  
        time.sleep(4)
```

Función para comprobar la confianza actual => check_trust():

Verifica si el nivel de confianza (trust_level) actual del usuario es suficiente para desbloquear partes de la historia. Esta función recorre la lista story_parts, la cual contiene fragmentos de la historia junto con el nivel de confianza necesario para desbloquearlos. Si el nivel de confianza es igual o mayor al requerido para un fragmento de historia, este se revela al usuario llamando a la función reveal_story(part).

```
def check_trust():  
    global trust_level  
    for part in story_parts:  
        if trust_level >= part["trust_required"]:  
            reveal_story(part)  
            story_parts.remove(part)
```


Función para mostrar mensaje de cierre => reveal_closing_message():

Imprime varias líneas de texto con un retraso para crear un efecto dramático antes de cerrar el programa cuando el nivel de frustración es crítico. Dichas líneas mostraran la decepción del lenguaje hacia el usuario.

```
def reveal_closing_message():
    closing_lines = [
        "closing lines"
    ]
    for line in closing_lines:
        print(line)
        time.sleep(4)
    sys.exit()
```

⚠ Para más información sobre el funcionamiento del código, consulte el README.

CICLO PRINCIPAL

Este ciclo se ejecuta indefinidamente, permitiendo que el usuario introduzca comandos de manera continua hasta que el programa se cierre. Al inicio de cada iteración, se verifica si el nivel de frustración ha superado el valor 15. Si es así, se ejecuta la función reveal_closing_message() que finaliza el programa.

PRINT

Si el comando comienza con print, se asume que el usuario quiere imprimir una expresión. La parte después de print se evalúa con eval(), que interpreta y ejecuta la expresión. Si la evaluación es exitosa, el resultado se imprime y el nivel de confianza (trust_level) aumenta en 1. A continuación, se llama a check_trust() para verificar si el nivel de confianza es suficiente para revelar más de la historia. Si ocurre un error (por ejemplo, si la expresión es inválida), se incrementa el nivel de frustración con increase_frustration().

Para imprimir el valor de una variable, puedes usar el comando print. Hay dos formas de hacerlo:

- >> **print** x
- >> **print**(x)

ASIGNACIÓN DE VARIABLES

Si el comando contiene un signo igual =, se asume que es una asignación de variables. Se ejecuta el comando completo usando `exec()`, lo que permite asignaciones dinámicas de variables. Si la asignación se realiza correctamente, el nivel de confianza aumenta y se verifica si se puede revelar más de la historia. Si hay un error en la asignación (por ejemplo, una expresión no válida), se maneja de la misma manera incrementando la frustración.

Puedes asignar variables usando el operador =. Por ejemplo:

- `>> x = 5`

IF CONDICIONAL

Si el comando comienza con `if`, se procesa como una condición. El comando se divide en dos partes: la condición (antes del `print`) y lo que debe imprimirse si la condición es verdadera (después del `print`). La condición se evalúa con `eval()`. Si es verdadera, se ejecuta la instrucción y se imprime el resultado. Si el bloque condicional se ejecuta correctamente, el nivel de confianza aumenta, y se intenta revelar más de la historia. Si hay algún error, el programa incrementa el nivel de frustración.

Puedes usar una estructura condicional `if` para ejecutar comandos solo si una condición es verdadera. Por ejemplo:

- `>> if x == 5 print("x es igual a 5")`

WHILE

El comando `while` se utiliza para crear un bucle condicional. La condición se encuentra antes del `do` y la expresión que se ejecuta repetidamente está después del `do`. La condición se evalúa usando `eval()` y, mientras sea verdadera, la expresión se ejecuta repetidamente con `exec()`. Si el bucle se ejecuta correctamente, el nivel de confianza aumenta, y el programa puede revelar más de la historia. Si ocurre algún error en la evaluación o en la ejecución, el nivel de frustración aumenta.

El bucle `while` se usa para repetir una operación mientras una condición sea verdadera. Debes usar la sintaxis `while <condición> do <expresión>`:

- `>> x = 0`
- **Assigned:** `x = 0`
- `>> while x < 5 do x += 1`



COMANDO NO VÁLIDO

Si el comando no es un print, una asignación, un if, o un while, se considera inválido. En ese caso, se imprime el mensaje "Unrecognized command" y se incrementa el nivel de frustración.

⚠ **Para más información sobre el funcionamiento del código, consulte el README.**

GESTIÓN DE ERRORES

Asignación incorrecta de variables en bucles while: El programa interpretaba mal las asignaciones dentro de los bucles. Solución: Se modificó la lógica de asignación para separar correctamente los comandos de control de flujo y las asignaciones.

Dificultad para bloquear la entrada del usuario durante la revelación de la historia: No se podía impedir que el usuario escribiera durante esta fase. Solución: Se añadieron mensajes de espera y se ignoraron entradas hasta que la historia se completara.

Manejo de errores impreciso: Algunos errores no proporcionaban suficiente información. Solución: Se mejoraron los mensajes de error para hacerlos más descriptivos y facilitar la corrección.

Frustración del script demasiado alta o baja: El programa podía terminar abruptamente o no reaccionar al mal uso de comandos. Solución: Ajuste del sistema de frustración para hacerlo más equilibrado.

Problemas con comandos encadenados: Los comandos separados por ; no se ejecutaban correctamente. Solución: Se implementó un manejo más robusto de los comandos múltiples en una misma línea.



BIBLIOGRAFÍA

CodePulse. (2018, 4 diciembre). Make YOUR OWN Programming Language - EP 1 -

Lexer [Vídeo]. YouTube. <https://www.youtube.com/watch?v=Eythq9848Fg>

CppNow. (2017, 15 junio). C++Now 2017: Mark Zeren “Esolangs” [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=d9bbQbPvxXE>

Esolang, the esoteric programming languages wiki. (s. f.).

https://esolangs.org/wiki/Main_Page

Hillel Wayne. (2021, 17 mayo). A Brief Introduction to Esoteric Programming

Languages [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=cQ7bcCrJMHc>

Wikipedia contributors. (2024, 24 septiembre). Esoteric programming language.

Wikipedia. https://en.wikipedia.org/wiki/Esoteric_programming_language