



UNIVERSIDAD ESTATAL A DISTANCIA
VICERRECTORÍA ACADÉMICA
ESCUELA DE CIENCIAS EXACTAS Y NATURALES
Cátedra Tecnología de Sistemas



Compiladores

Código: [03307]

Proyecto 1. Valor 1%

Temas de Estudio

- Tema 1. Introducción a los compiladores
- Tema 2. Un traductor simple orientado a la sintaxis

Objetivo

Aplicar los conocimientos adquiridos en el curso, centrándonos en la programación relacionada con los conceptos de compiladores, intérpretes, para validar a nivel de análisis léxico y la sintaxis, de un código fuente el cual es formado por entrada de cadenas de textos.

Software de Desarrollo

El proyecto debe ser realizado en el lenguaje de programación Java, utilizando como IDE NetBeans en modo carácter. No se debe usar el modo gráfico.

Desarrollo

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python se utiliza ampliamente en diversos campos como desarrollo web, análisis de datos, inteligencia artificial, scripting, automatización, y más.

1. El objetivo de este primero proyecto, es realizar un análisis que identifique los tokens o componentes léxicos basado en código de Python, para verificar su correcto uso y estructuras sintácticas, según las reglas que los determinan en la rúbrica no el lenguaje, es decir, respetamos las indicaciones del proyecto.
2. El programa interprete que usted va a realizar en NetBeans con el lenguaje java, debe ser independiente de una ruta física. Esto implica que el programa pueda ejecutarse en cualquier máquina y en cualquier directorio sin depender de configuraciones específicas, para ello se hará uso de la consola de windows (CMD), para ejecutar el archivo compilado (.jar)
3. En este proyecto, se llevará a cabo la lectura de un archivo con extensión .py que contiene un ejemplo de código en Python.
4. Para que este archivo .py pueda ser leído, no deberá depender de un nombre específico ni de una ruta o carpeta estática, ni de configuraciones propias del desarrollador, por lo contrario, deberá estar en la misma carpeta que se encuentre el archivo compilado .jar

5. Se debe detectar y registrar los posibles errores en un archivo de salida con extensión `.log`, las especificaciones de este punto detallan en la rúbrica.
6. No se permite el uso de librerías de terceros, que son fragmentos de código desarrollados por otras personas y encapsulados en componentes que podrían utilizarse como referencia en la solución del problema a través del código fuente.

Código fuente:

- El código que se muestra a continuación, corresponde al juego tradicional “**El Gato**” y está desarrollado en el lenguaje programación Python, e implementa la lógica para jugar contra la computadora (CPU).
- El código no tiene ningún error y lo puede ejecutar online en el siguiente sitio que interpreta código python <https://www.programiz.com/python-programming/online-compiler/>
- Usted puede trabajar a partir de dicho código para realizar lo que se le solicita, pero tenga en cuenta que el código, será modificado por el tutor (a) para agregar errores según se mencionan en la rúbrica para poder comprobar si el estudiante realizó la validación que se le solicitó.
- A continuación, se especifica el código con el que se estará trabajando en este proyecto, por favor tome en consideración que dicho código está disponible en un archivo descargable en la plataforma “Moodle” con la extensión `.py`

```
import random

def imprimir_tablero(tablero):
    """ Función para imprimir el tablero actual """
    for i in range(3):
        for j in range(3):
            print(f" {tablero[i][j]} ", end="")
            if j < 2:
                print("|", end="")
        print() # Salto de línea después de cada fila
        if i < 2:
            print("-----") # Línea horizontal entre filas

def verificar_ganador(tablero, jugador):
    """ Función para verificar si hay un ganador """
    # Verificar filas, columnas y diagonales
    for i in range(3):
        if (all(tablero[i][j] == jugador for j in range(3)) or          # Filas
            all(tablero[j][i] == jugador for j in range(3)) or          # Columnas
            all(tablero[i][i] == jugador for i in range(3)) or          # Diagonal principal
            all(tablero[i][2 - i] == jugador for i in range(3))):      # Diagonal secundaria
            return True
    return False

def movimiento_cpu(tablero, jugador):
    """ Función para que la CPU realice un movimiento """
    # Estrategia sencilla: mover aleatoriamente en una casilla vacía
    movimientos_posibles = [(i, j) for i in range(3) for j in range(3) if tablero[i][j] == " "]
    fila, columna = random.choice(movimientos_posibles)
    tablero[fila][columna] = jugador

def juego_gato():
    """ Función principal para ejecutar el juego """
    tablero = [{" " for _ in range(3)] for _ in range(3)]
```

```
# Solicitar nombre y edad del jugador
nombre = input("Ingrese su nombre: ")
while True:
    edad_str = input("Ingrese su edad: ")
    try:
        edad = int(edad_str)
        break # Si la conversión es exitosa, salir del bucle
    except ValueError:
        print("Error: La edad debe ser un número entero.")

jugador_usuario = nombre
jugador_cpu = "CPU"
jugador_actual = "X" # Empieza el usuario

resultado = None

while True:
    # Verificar si ya hay un resultado antes de imprimir el tablero
    if resultado:
        break

    imprimir_tablero(tablero)
    print(f"\nTurno del jugador {jugador_usuario if jugador_actual == 'X' else jugador_cpu} ({jugador_actual})")

    if jugador_actual == "X": # Turno del usuario
        try:
            fila, columna = map(int, input("Ingrese fila y columna (ej. '0 0'): ").split())
        except ValueError:
            print("Error: Por favor ingrese dos números separados por espacio.")
            continue

        if not (0 <= fila < 3 and 0 <= columna < 3):
            print("Error: Coordenadas fuera de rango. Debe ser entre 0 y 2.")
            continue

        if tablero[fila][columna] != " ":
            print("Casilla ocupada. Intente de nuevo.")
            continue

        tablero[fila][columna] = jugador_actual
    else: # Turno de la CPU
        movimiento_cpu(tablero, jugador_actual)

    if verificar_ganador(tablero, jugador_actual):
        resultado = f"\nEl jugador {jugador_usuario if jugador_actual == 'X' else jugador_cpu} ({jugador_actual}) ha ganado!"
        break

    if all(tablero[i][j] != " " for i in range(3) for j in range(3)):
        resultado = "\nEmpate!"
        break

    jugador_actual = "O" if jugador_actual == "X" else "X"

# Mostrar el tablero final y el resultado
imprimir_tablero(tablero)
print(resultado)
juego_gato()
```


Evaluación:

#	Requerimientos a evaluar
1	<p>Lectura archivo: se debe leer un archivo de código Python con extensión .py no debe leer otra extensión que no sea la indicada, <u>de lo contrario no está cumpliendo con este punto.</u></p> <ul style="list-style-type: none"> La ejecución del ejecutable .jar debe poder realizarse desde la consola CMD o símbolo del sistema. En la plataforma se provee un ejemplo en dicho formato de extensión indicado, debe utilizar ese. Al leer el archivo su programa debe aceptar espacios en blanco, líneas en blanco, cualquier espacio en general que se encuentre y que se distinga entre mayúsculas y minúsculas. Es decir, el formato original del archivo debe ser respetado en su totalidad. La lectura del archivo debe ser cargada en la memoria del programa para realizar los diferentes análisis y verificaciones propuestos en este proyecto. Es importante, que el archivo .py, no dependa de un nombre en específico para poder ser leído, y que no dependa de una ruta estática o física ni configuraciones personalizadas por el desarrollador (estudiante), de lo contrario no está cumpliendo con este punto. Donde se encuentre el archivo compilado de su programa .jar en la misma carpeta esté el archivo .py que se va a leer. Por lo anterior la ejecución de su programa deberá realizarse de la siguiente manera: Java -jar suPrograma.jar ProgramaFuente.py
2	<p>Manejo de errores: A partir del archivo que se leyó se debe crear otro archivo con extensión .log</p> <ul style="list-style-type: none"> El nombre resultante del nuevo archivo debe respetar el nombre del archivo original y agregarle un guion con la descripción “-errores.log” el archivo debe verse de esta manera: “NombreArchivo-errores.log” donde “nombreArchivo” depende del nombre que tenga dicho archivo de entrada .py Se debe respetar el contenido y formato del archivo original .py, únicamente, deberá agregarle 3 dígitos al inicio de la línea seguidos del texto, por ejemplo: <pre>001 import random 002 003 def imprimir_tablero(tablero): 004 """ Función para imprimir el tablero actual """</pre> <ul style="list-style-type: none"> En los próximos puntos a evaluar de esta rúbrica, en caso que se encuentre errores, se deberá indicar el error utilizando el siguiente formato: <ul style="list-style-type: none"> <u>Número del error:</u> la tipificación es a criterio del desarrollador <u>Descripción:</u> corresponde a un error particular según lo que debió hacerse Se debe ver de la siguiente manera: Error 200. El import no se encuentra al inicio del código Otra manera de agregar los errores es al final del archivo, pero debe de indicar el número de línea donde encontró el error y respetar lo anterior, ejemplo: Error 200. Línea 20. El import no se encuentra al inicio del código

- Considere que en una misma línea puede haber más de un error que debe detectar.
- Los errores deben ser específicos y claros indicando con exactitud el problema encontrado.
- Por cada error encontrado se debe crear una línea de error, es decir, si en la misma línea hay 3 errores, entonces deben aparecer tres líneas de error y cada una aclara el error de manera específica.

3 **Identificadores:** En Python, los identificadores son los nombres que se utilizan para identificar variables, funciones, clases, módulos y otros objetos. Para este proyecto vamos solo a verificar la correctitud de las variables. Para definir una variable en Python se asigna un valor a un nombre de variable usando el operador de asignación =

Reglas para los identificadores en Python:

- ✓ Comienzo del identificador:
 - Debe comenzar con una letra (a-z, A-Z) o un guion bajo (_)
 - No puede comenzar con un número.
 - No puede comenzar con caracteres especiales
- ✓ Caracteres permitidos:
 - Después del primer carácter, los identificadores pueden contener letras (a-z, A-Z), dígitos (0-9) y guiones bajos (_).
- ✓ Palabras reservadas:
 - No se pueden usar palabras reservadas de Python como identificadores. Estas palabras tienen un significado especial en el lenguaje, para este proyecto vamos a considerar  validar solo estas palabras reservadas:

import	break	elif	or
def	continue	False	pass
for	else	finally	raise
if	and	from	True
print	as	global	with
return	assert	is	yield
in	async	lambda	not
while	await	None	del
try	class	nonlocal	except
range	input		

- ✓ Sensibilidad a mayúsculas y minúsculas:
 - Python distingue entre mayúsculas y minúsculas, por lo que variable, Variable y VARIABLE serían identificadores diferentes.

Ejemplos de identificadores válidos:

- ✓ variable1 = 10
- ✓ _variable = "Hola"
- ✓ mi_variable = 3.14
- ✓ MiVariable = 5

	<ul style="list-style-type: none"> ✓ Nombre = input("Ingrese su nombre: ") ✓ edad_str = int(edad_str) ✓ jugador_actual = "X" ✓ resultado = None <p>Ejemplos de identificadores no válidos:</p> <ul style="list-style-type: none"> ✓ 1variable = 10 # No puede comenzar con un número ✓ mi-variable = "Hola" # No puede contener guiones ✓ class = "Clase" # 'class' es una palabra reservada
4	<p>Import: En Python, aunque se permite técnicamente colocar las declaraciones <u>import</u> en cualquier parte del código, la práctica establecida es colocarlas al comienzo del archivo. Esta convención mejora la claridad y la legibilidad del código, facilitando a otros desarrolladores la identificación rápida de las dependencias del script. Basándonos en esta directriz, todas las declaraciones <u>import</u> deberán ubicarse al inicio del código en este proyecto. Cualquier declaración <u>import</u> encontrada en una posición diferente será considerada incorrecta para los fines de esta evaluación.</p>
5	<p>Comentarios: En Python, hay dos formas principales de añadir comentarios en el código: comentarios de una línea y comentarios de varias líneas o también conocidos como docstrings, para nuestro proyecto <u>solo vamos a considerar los comentarios de una línea</u>, y se requiere que se contabilice cuantos comentarios hay en el código que se está evaluando, es importante tomar en consideración esta regla, los comentarios de una línea comienzan con el símbolo # y pueden aparecer en cualquier lugar de la línea de código después de un espacio en blanco o código activo. Estos comentarios son útiles para agregar explicaciones breves o aclaraciones rápidas sobre el código. Algunos ejemplos:</p> <pre>def verificar_ganador(tablero, jugador): """ Función para verificar si hay un ganador """ ➡ # Verificar filas, columnas y diagonales for i in range(3): if (all(tablero[i][j] == jugador for j in range(3)) or all(tablero[j][i] == jugador for j in range(3)) or # Filas # Columnas</pre> <p>Al final del archivo de errores deben indicarse la cantidad de comentarios encontrados, por ejemplo:</p> <ul style="list-style-type: none"> • 5 líneas de comentario
6	<p>Token de operadores comparación: En Python, los tokens de símbolos son caracteres especiales y secuencias que tienen significados específicos en el lenguaje, entre ellos los operadores de comparación determinados por los siguientes símbolos:</p> <ul style="list-style-type: none"> • == (igual a) • != (no igual a) • < (menor que) • > (mayor que) • <= (menor o igual que)

- `>=` (mayor o igual que)

Se requiere contabilizar la cantidad de tokens de operadores de comparación, el resultado debe quedar documentado considerando, cantidad de tokens encontrados, luego la palabra token y por último el símbolo, deben quedar al final del archivo de errores, y debe verse de la siguiente manera:

- 6 Token ==
- 2 Token <
- 2 Token <=

7 **input:** El comando `input()` en Python se utiliza para obtener entrada del usuario desde la consola. La sintaxis básica es:

- `variable = input(prompt)`

prompt: Es una cadena que se muestra al usuario como mensaje para solicitar la entrada. No es necesario usar el parámetro `prompt`, pero es útil para indicar al usuario qué tipo de entrada se espera, a continuación, un ejemplo con `prompt`:

- `nombre = input("Ingrese su nombre: ")`

De acuerdo al ejemplo anterior, la sintaxis que debe validar es la siguiente:

- Variable
- Símbolo =
- Comando `input`
- Abre y cierra paréntesis
- Y comillas dobles dentro de los paréntesis después del paréntesis de apertura y antes del paréntesis de cierre, lo que está dentro de las comillas no se debe validar.

Nota: se aclara que en el lenguaje de python existen otras maneras del uso del comando `INPUT`, pero para esta evaluación, apegarse a la sintaxis indicada.

8 **Documentar:** Documente en un archivo de texto con extensión `.txt` llamado `README.txt` los pasos necesarios para ejecutar su programa, junto con cualquier información adicional que desee clarificar al profesor sobre su proyecto. Asegúrese de verificar y confirmar que estos pasos funcionen correctamente antes de la entrega.

Honestidad Académica



<https://audiovisuales.uned.ac.cr/play/player/23048>

Nota Importante

Cada estudiante es responsable del contenido que entrega, si no es el archivo correcto, no podrá entregarlo posterior a la fecha establecida.

Si el contenido del archivo coincide con algún otro estudiante, o se comprueba que no es de su autoría, se aplicaría lo indicado en la plataforma en el documento [Lineamientos ante casos de plagio](#)

Indicaciones Importantes

- Es obligatorio que incluya todo el directorio donde se encuentra el proyecto 1.
- El proyecto 1 debe estar desarrollado en Netbeans que es la herramienta oficial del curso.
- El programa debe ser modular, utilizando de la mejor manera funciones definidas por usted.
- Los trabajos deben realizarse en forma individual. Dentro del código del programa debe de indicar la documentación que explique cómo fue realizado el programa.
- Si utiliza código de algún ejemplo del libro, o de otra fuente que no sea de su autoría, debe de indicarlo.
- Comprima todos los archivos en un solo archivo .zip o .rar.
- Nombre del archivo que envía: debe ser nombre y primer apellido del estudiante, y nombre del proyecto. Ejemplo: JuanRojas-proyecto1.
- La entrega del proyecto 1 se debe realizar en las fechas establecidas en la plataforma de aprendizaje en línea Moodle en el apartado que se indique.
- Si no concluyó a tiempo la tarea, debe entregar lo que pudo hacer e incluir una carta explicando las razones por las cuales no finalizó.

Rúbrica de Evaluación

Criterio	Cumple a satisfacción lo indicado en la evaluación	Cumple medianamente en lo indicado en la evaluación	Cumple en contenido y formato, pero los aportes no son significantes	No cumple o no presenta lo solicitado
1. Lectura del archivo: Al leer el archivo el programa acepte espacios en blanco, líneas en blanco, cualquier espacio en general que se encuentre y que se distinga entre mayúsculas y minúsculas. Es decir, el formato original del archivo debe ser respetado en su totalidad. La ejecución del ejecutable .jar debe poder realizarse desde la consola CMD o símbolo del sistema.	5	2	1	0
2. Lectura del archivo: El archivo .py, no dependa de un nombre en específico para poder ser leído, y que no dependa de una ruta estática o física ni configuraciones personalizadas por el desarrollo	10	5	3	0
3. Lectura del archivo: Donde se encuentre el archivo compilado de su programa .jar en la misma carpeta esté el archivo .py que se va a leer.	5	2	1	0
4. Para el manejo de errores: A partir del archivo que se leyó se debe crear otro archivo con extensión .log	5	2	1	0
5. Para el manejo de errores: El nombre resultante del nuevo archivo debe respetar el nombre del archivo original y agregarle un guion con la descripción "-errores.log" el archivo debe verse de esta manera: "NombreArchivo-errores.log" donde "nombreArchivo" depende del nombre que tenga dicho archivo de entrada .py	5	2	1	0
6. Para el manejo de errores: Se debe respetar el contenido y formato del archivo original .py sin embargo, deberá agregarle 3 dígitos al inicio de la línea seguidos del texto	5	3	1	0

<p>7. Para el manejo de errores: Se deberá indicar el error utilizando el siguiente formato:</p> <ul style="list-style-type: none"> - Número del error: la tipificación es a criterio del desarrollador - Descripción: corresponde a un error particular según lo que debió hacerse <p>Se debe ver de la siguiente manera: Error 200. El import no se encuentra al inicio del código Otra manera de agregar los errores es al final del archivo, pero debe de indicar el número de línea donde encontró el error y respetar lo anterior, ejemplo: Error 200. Línea 20. El import no se encuentra al inicio del código. Por cada error encontrado se debe generar una línea de error.</p>	10	5	3	0
8. Según las reglas de los identificadores, verificar la correctitud de las variables. Para definir una variable en Python se asigna un valor a un nombre de variable usando el operador de asignación =	10	5	2	
9. Import: Cualquier declaración import encontrada en una posición diferente al inicio del código, será considerada incorrecta para los fines de esta evaluación	10	5	2	0
10. Comentarios: se requiere que se contabilice cuantos comentarios hay en el código que se está evaluando, es importante tomar en consideración esta regla, los comentarios de una línea comienzan con el símbolo # y pueden aparecer en cualquier lugar de la línea de código después de un espacio en blanco o código activo. La cantidad total de comentarios se muestra al final del archivo de errores.log.	5	3	2	0
11. Token de operadores comparación: Se requiere contabilizar la cantidad de tokens de operadores de comparación, el resultado debe quedar documentado en el mismo archivo de errores al final	5	2	1	
12. Token de operadores comparación: Debe respetar el formato solicitado para escribir en el archivo de errores los tokens encontrados, indicando primero la cantidad de tokens encontrado, luego la palabra token y por último el símbolo	5	2	1	0

13. Función INPUT: válida la sintaxis del uso correcto de la función input. -Variable -Símbolo = -Comando input -Abre y cierra paréntesis - Y comillas dobles dentro de los paréntesis después del paréntesis de apertura y antes del paréntesis de cierre, lo que está dentro de las comillas no se debe validar.	15	10	5	0
14. Documentar: Documente en un archivo de texto con extensión .txt llamado README.txt los pasos necesarios para ejecutar su programa, junto con cualquier información adicional que desee clarificar al profesor sobre su proyecto. Asegúrese de verificar y confirmar que estos pasos funcionen correctamente antes de la entrega	5	2	1	0
TOTAL	100	50	25	0