VNIVERSITAT DE VALÈNCIA

Laboratorio de ESTRUCTURAS DE DATOS Y ALGORITMOS
Grado en Ciencia de Datos (1º)
Curso 2024-25

Práctica Nº 4: Programación con Pilas y Colas

Periodo de realización: Semana del **31/03 a 04/04/2025**

Metodología de trabajo

Fases	Tipo	Dedicación
1. Resolver las tareas planteadas en este guion antes de iniciar la sesión presencial en el laboratorio.	No presencial	Máx. 4,5 h.
2. Resolver en el laboratorio un nuevo ejercicio basado en la resolución de las tareas previas.	Presencial	Máx. 3 h.

Introducción

En esta práctica se aplicarán Pilas y Colas para resolver un problema relacionado con el análisis y extracción de información de textos. En particular, se analizará el texto de un archivo que contiene código Python, pero la idea sirve para ver cuáles pueden ser los mecanismos que permitirían analizar textos escritos utilizando un lenguaje o un formato diferente. Es el caso de los archivos de intercambio de información que utilizan lenguajes de marcas (etiquetas), como XML, JSON, etc.

Para esta práctica se toma como referencia una función vista en clase de teoría, que permite analizar la concordancia de paréntesis en una expresión algebraica, y se extiende su aplicación a otro tipo de textos. Para ello, se deben resolver las tareas indicadas a continuación, que como siempre servirán como referencia para resolver un nuevo ejercicio durante la sesión de laboratorio.

En el Aula Virtual se proporcionan los siguientes archivos Python que deben ser utilizados en la resolución de las tareas:

- `pr4_parentesis.py`: Incluye la función que analiza la concordancia de paréntesis.
- `clase_pila.py`: Implementación de la clase *Pila*.
- `clase_cola.py`: Implementación de la clase *Cola*.
- `test.py`: Archivo para utilizar como entrada de datos en las pruebas de las tareas.

Ejercicios (Fase 1)

Tarea 1

La función `AnalizarParentesis`, incluida en el archivo "`pr4_parentesis.py`", analiza la corrección del uso de paréntesis "`(`" y "`)`" en una cadena de texto. Este tipo de análisis tiene especial interés para realizarlo para cadenas que representan expresiones algebraicas, pero los paréntesis no solo se utilizan en estos casos, ya que son una herramienta de los lenguajes (naturales o de programación) que tienen aplicación en otros contextos.

En particular, en el lenguaje Python los paréntesis también se utilizan para:

- Delimitar el conjunto de argumentos de una función, tanto en su definición como en las llamadas.
- Definir los valores de una tupla.

La función `AnalizarParentesis` permite analizar cualquier cadena de caracteres, no importa cuál sea su contenido. Por eso vamos a utilizarla para analizar todo un programa Python.
Esta primera tarea consiste en escribir un programa, "`pr4_fase1.py`", que lea las líneas de un archivo que contiene código Python y que valide la concordancia de paréntesis de cada una de estas líneas. Cuando el programa detecte una línea en la que el uso de paréntesis sea incorrecto deberá mostrar por pantalla el número de línea correspondiente y el listado de errores encontrados. Para hacer este programa copia en el mismo programa la función `AnalizarParentesis`, no la importes porque en posteriores tareas vas a modificarla y es preferible hacer todos los cambios en el mismo archivo.

Si se ejecuta el programa usando como entrada el archivo "`test.py`" el listado de errores es:

```
Línea 48
'(' no cerrado en 28
Línea 49
')' no abierto en 48
Línea 59
'(' no cerrado en 28
Línea 60
')' no abierto en 59
Línea 65
'(' no cerrado en 27
Línea 66
')' no abierto en 70
```

Tarea 2

Se debe modificar ahora la función `AnalizarParentesis` (que has copiado en el programa) para que devuelva también un valor que indique si la cadena pasada como argumento contiene paréntesis o no (además de la lista de errores que ya devuelve).

Una vez realizado este cambio, se debe modificar el programa de la tarea 1 para que almacene en una cola (para respetar el orden de aparición) los números de línea del archivo que contengan algún paréntesis, "`(`" y/o "`)`". No importa que tengan errores o no. Al finalizar la lectura y análisis del archivo se deberá mostrar por pantalla el contenido de la cola generada.

Si se ejecuta el programa usando como entrada el archivo "`test.py`" la cola de líneas con paréntesis será:

```
Líneas con paréntesis:

19 23 26 29 32 35 36 40 42 44 45 46 48 49 50 52 54 55 56 57 59 60 63 64 65 66 67 68 70 71 72 74
75 78 79 80 84 87 88 89 90 93 95 97 98 103 104 105 108 110 113 115 119
```

Tarea 3

Para ser coherente con la forma de actuar en la tarea 2, se debe modificar el programa para que las líneas con errores detectadas en la tarea 1 también sean almacenadas por el programa en una cola (para respetar el orden de aparición), que será mostrada tras leer y analizar todo el archivo. Para ello, es necesario definir en el programa una clase de objetos para representar las líneas con errores. Esta clase se llamará `Linea` y un objeto de esta clase tendrá como datos el *número de línea* dentro del archivo y la *lista con los errores* encontrados en ella. Estos dos datos deberán ser proporcionados directamente como argumentos del **constructor** de la clase. También se deberá sobrecargar el operador **`__str__`** para que se pueda obtener la versión texto de estos objetos.

Al detectar una línea con error se generará un objeto de tipo `Linea` con los datos adecuados y se guardará en la cola.

Al finalizar las tres tareas, el programa deberá comportarse de la siguiente manera:

1) Para cada línea del archivo deberá:
   - Analizar la concordancia de paréntesis.
   - Si la línea contiene paréntesis la guardará el número de línea en una cola.
   - Si la línea contiene paréntesis y errores de concordancia también guardará un objeto de la clase *`Linea`*, correctamente formado, en una segunda cola solo para líneas con errores.
2) Al finalizar la lectura mostrará tanto la cola de paréntesis como la cola de errores.

Si se ejecuta el programa usando como entrada el archivo "`test.py`" la salida por pantalla será:

(página siguiente …)

```
Resultados
==========
* Líneas con paréntesis:
19 23 26 29 32 35 36 40 42 44 45 46 48 49 50 52 54 55 56 57 59 60 63 64 65 66 67 68 70 71 72 74
75 78 79 80 84 87 88 89 90 93 95 97 98 103 104 105 108 110 113 115 119

* Líneas con errores:
Línea 48
'(' no cerrado en 28
Línea 49
')' no abierto en 48
Línea 59
'(' no cerrado en 28
Línea 60
')' no abierto en 59
Línea 65
'(' no cerrado en 27
Línea 66
')' no abierto en 70
```

Nota: Observa que en la cola de líneas con paréntesis están incluidas las líneas con errores.

Fase 2: Tarea final

Resolución de un nuevo ejercicio planteado durante la sesión presencial en el laboratorio.