Introducción a la Programación

Guía Práctica 8
Pilas, Colas, Diccionarios y Archivos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1. Pilas

Ejercicio 1. Implementar una solución para el siguiente problema.

`problema generar_nros_al_azar` (in cantidad: $\mathbb{Z}$, in desde: $\mathbb{Z}$, in hasta: $\mathbb{Z}$) : Pila[$\mathbb{Z}$] {
    `requiere:` {$cantidad \geq 0$}
    `requiere:` {$desde \leq hasta$}
    `asegura:` {El tamaño de *res* es igual a *cantidad*}
    `asegura:` {Todos los elementos de *res* son valores entre *desde* y *hasta* (ambos inclusive), seleccionados aleatoriamente con probabilidad uniforme}
}

Para generar números en un rango con probabilidad uniforme, pueden usar la función `random.randint`(< `desde` >, < `hasta` >) que devuelve un número en el rango indicado. Recuerden importar el módulo random con `import random`. Además, pueden usar la clase `LifoQueue`() que es un ejemplo de una implementación básica de una pila:

```
from queue import LifoQueue as Pila #importa LifoQueue y le asigna el alias Pila

p = Pila() #crea una pila
p.put(1) # apila un 1
elemento = p.get() # desapila
p.empty() # devuelve true si y solo si la pila está vacía
```

Ejercicio 2. Implementar una solución para el siguiente problema.

`problema cantidad_elementos` (in p: Pila) : $\mathbb{Z}$ {
    `requiere:` {True}
    `asegura:` {*res* es igual a la cantidad de elementos que contiene *p*}
}

No se puede utilizar la función `LifoQueue.qsize()`. Tener en cuenta que, al usar `get()` para recorrer la pila, se modifica el parámetro de entrada, ya que los elementos se eliminan al accederse. Dado que la especificación lo define como de tipo `in`, debe restaurarse posteriormente.

Ejercicio 3. Implementar una solución para el siguiente problema.

`problema buscar_el_maximo` (in p: Pila[$\mathbb{Z}$]) : $\mathbb{Z}$ {
    `requiere:` {*p* no está vacía}
    `asegura:` {*res* es un elemento de *p*}
    `asegura:` {*res* es mayor o igual a todos los elementos de *p*}
}

Ejercicio 4. Implementar una solución para el siguiente problema.

`problema buscar_nota_maxima` (in p: Pila[$seq\langle Char\rangle \times \mathbb{Z}$]) : $seq\langle Char\rangle \times \mathbb{Z}$ {
    `requiere:` {*p* no está vacía}
    `requiere:` {los elementos de *p* no tienen valores repetidos en la segunda posición de las tuplas}
    `asegura:` {*res* es una tupla de *p*}
    `asegura:` {No hay ningún elemento en *p* cuya segunda componente sea mayor que la segunda componente de *res* }
}

Ejercicio 5. Implementar una solución, que use pila, para el siguiente problema.

`problema esta_bien_balanceada` (in s: $seq\langle Char \rangle$) : Bool {
    `requiere:` {s solo puede tener números enteros, espacios y los símbolos `'(', ')', '+', '-', '*', '/'`}
    `asegura:` {res = true $\leftrightarrow$ (La cantidad de paréntesis de apertura '(és igual a la de cierre ')') y (Para todo prefijo de 's', la cantidad de paréntesis de cierre no supera a la de apertura)}
}

Por cada paréntesis de cierre debe haber uno de apertura correspondiente antes de él. Las fórmulas pueden tener:

- números enteros
- operaciones básicas $+$, $-$, $*$ y $/$
- paréntesis
- espacios

Entonces las siguientes son fórmulas aritméticas con sus paréntesis bien balanceados:

$1 + (2 \text{ x } 3 - (20 / 5))$
$10 * (1 + (2 * (-1)))$

Y la siguiente es una fórmula que no tiene los paréntesis bien balanceados:

$1 +) \ 2 \text{ x } 3 \ (\ ()$

Ejercicio 6. La notación polaca inversa, también conocida como notación postfix, es una forma de escribir expresiones matemáticas en la que los operadores siguen a sus operandos. Por ejemplo, la expresión "3 + 4" se escribe como "3 4 +" en notación postfix. Para evaluar una expresión en notación postfix, se puede usar una pila. Implementar una solución para el siguiente problema.

`problema evaluar_expresion` (in s: $seq\langle Char \rangle$) : $\mathbb{R}$ {
    `requiere:` {s solo contiene números enteros y los operadores binarios `+`, `-`, `*` y `/`}
    `requiere:` {Todos los elementos (operandos y operadores) están separados por un único espacio}
    `requiere:` {La expresión es sintácticamente válida: cada operador binario tiene exactamente dos operandos previos disponibles en el momento de su evaluación.}
    `asegura:` {res es el valor obtenido al evaluar la expresión postfija representada por s}
}

Para resolver este problema, se recomienda seguir el siguiente algoritmo:

1. Dividir la expresión en tokens (operandos y operadores) utilizando espacios como delimitadores.
2. Recorre los tokens uno por uno.
    a) Si es un operando, agrégalo a una pila.
    b) Si es un operador, saca los dos operandos superiores de la pila, aplícale el operador y luego coloca el resultado en la pila.
3. Al final de la evaluación, la pila contendrá el resultado de la expresión.

Ejemplo de uso:

```
expresion = "3 4 + 5 * 2 -"
resultado = evaluar_expresion(expresion)
print(resultado)  # Debería imprimir 33
```

Ejercicio 7. Implementar una solución para el siguiente problema.

`problema intercalar` (in p1: Pila, in p2: Pila) : Pila {
    `requiere:` {$p1$ y $p2$ tienen la misma cantidad de elementos}
    `asegura:` {res solo contiene los elementos de $p1$ y $p2$}
    `asegura:` {res contiene todos los elementos de $p1$ y $p2$, intercalados y respetando el orden original}
    `asegura:` {El tope de la pila res es el tope de $p2$}

`asegura:` {El tamaño de *res* es igual al doble del tamaño de $p1$}
}

Nota: Ojo que hay que recorrer dos veces para que queden en el orden apropiado al final.

2. Colas

Ejercicio 8. Implementar una solución para el siguiente problema.

`problema generar_nros_al_azar` (in cantidad: $\mathbb{Z}$, in desde: $\mathbb{Z}$, in hasta: $\mathbb{Z}$) : Cola[$\mathbb{Z}$] {
`requiere:` {$cantidad \geq 0$}
`requiere:` {$desde \leq hasta$}
`asegura:` {El tamaño de *res* es igual a *cantidad*}
`asegura:` {Todos los elementos de *res* son valores entre *desde* y *hasta* (ambos inclusive), seleccionados aleatoriamente con probabilidad uniforme}
}

Para generar números en un rango con probabilidad uniforme, pueden usar la función `random.randint`(< `desde` >, < `hasta` >) que devuelve un número en el rango indicado. Recuerden importar el módulo random con `import random`. Pueden usar la clase `Queue`() que es un ejemplo de una implementación básica de una Cola:

```
from queue import Queue as Cola #importa Queue y le asigna el alias Cola

c = Cola() #creo una cola
c.put(1) # encolo el 1
elemento = c.get() # desencolo
c.empty() # devuelve true si y solo si la cola está vacía
```

Ejercicio 9. Implementar una solución para el siguiente problema.

`problema cantidad_elementos` (in c: Cola) : $\mathbb{Z}$ {
`requiere:` {True}
`asegura:` {*res* es igual a la cantidad de elementos que contiene c}
}

No se puede utilizar la función `Queue.qsize()`.
Comparar el resultado con la implementación utilizando una pila en lugar de una cola.

Ejercicio 10. Implementar una solución para el siguiente problema.

`problema buscar_el_maximo` (in c: Cola[$\mathbb{Z}$]) : $\mathbb{Z}$ {
`requiere:` {c no está vacía}
`asegura:` {*res* es un elemento de c}
`asegura:` {*res* es mayor o igual a todos los elementos de c}
}

Comparar con la versión usando pila.

Ejercicio 11. Implementar una solución para el siguiente problema.

`problema buscar_nota_minima` (in c: Cola[$seq\langle Char \times \mathbb{Z} \rangle$]) : ($seq\langle Char \times \mathbb{Z} \rangle$) {
`requiere:` {c no está vacía}
`requiere:` {los elementos de c no tienen valores repetidos en la segunda componente de las tuplas}
`asegura:` {*res* es una tupla de c}
`asegura:` {No hay ningún elemento en c cuya segunda componente sea menor que la de *res* }
}

Ejercicio 12. Implementar una solución para el siguiente problema.

`problema intercalar` (in c1: Cola, in c2: Cola) : Cola {
    `requiere:` {$c1$ y $c2$ tienen la misma cantidad de elementos}
    `asegura:` {res solo contiene los elementos de $c1$ y $c2$}
    `asegura:` {res contiene todos los elementos de $c1$ y $c2$, intercalados y respetando el orden original}
    `asegura:` {El primer elemento de res es el primer elemento de $c1$}
    `asegura:` {El tamaño de res es igual al doble del tamaño de $c1$}
}

Ejercicio 13. Bingo: un cartón de bingo contiene 12 números al azar en el rango $[0, 99]$. Implementar una solución para cada uno de los siguientes problemas.

1. `problema armar_secuencia_de_bingo` () : Cola[$\mathbb{Z}$] {
       `requiere:` {True}
       `asegura:` {res solo contiene 100 números del 0 al 99 inclusive, sin repetidos}
       `asegura:` {Los números de res están ordenados al azar}
   }

   Para generar números pseudoaleatorios pueden usar la función `random.randint`(< `desde` >, < `hasta` >) que devuelve un número en el rango indicado. Recuerden importar el módulo random con `import random`.

2. `problema jugar_carton_de_bingo` (in carton: $seq\langle\mathbb{Z}\rangle$, in bolillero: Cola[$\mathbb{Z}$]) : $\mathbb{Z}$ {
       `requiere:` {$carton$ solo contiene 12 números, sin repetidos, con valores entre 0 y 99, ambos inclusive}
       `requiere:` {$bolillero$ solo contiene 100 números, ordenados al azar, del 0 al 99, ambos inclusive, sin repetidos}
       `asegura:` {res es la cantidad mínima de jugadas necesarias para que todos los números del $carton$ hayan salido del bolillero}
   }

Ejercicio 14. Vamos a modelar una guardia de un hospital usando una cola donde se van almacenando los pedidos de atención para los pacientes que van llegando. A cada paciente se le asigna una prioridad del 1 al 10 (donde la prioridad 1 es la más urgente y requiere atención inmediata) junto con su nombre y la especialidad médica que le corresponde. Implementar una solución para el siguiente problema.

`problema pacientes_urgentes` (in c:Cola[$\mathbb{Z}\times$ $seq\langle Char\rangle$ $\times$ $seq\langle Char\rangle$]) : $\mathbb{Z}$ {
    `requiere:` {Todos los elementos de c tienen como primer componente de la tupla un entero positivo y menor a 11}
    `asegura:` {res es la cantidad de elementos de c que tienen como primer componente de la tupla un número menor a 4}
}

Ejercicio 15. La gerencia de un banco nos pide modelar la atención de los clientes usando una cola donde se van registrando los pedidos de atención. Cada vez que ingresa una persona a la entidad, debe completar sus datos en una pantalla que está a la entrada: Nombre y Apellido, DNI, tipo de cuenta ($true$ si es preferencial o $false$ en el caso contrario) y si tiene prioridad ($true$ o $false$) por ser adulto +65, embarazada o con movilidad reducida.

La atención a los clientes se da por el siguiente orden: primero las personas que tienen prioridad, luego las que tienen cuenta bancaria preferencial y por último el resto. Dentro de cada subgrupo de clientes, se respeta el orden de llegada.

1. Dar una especificación para el problema planteado.
2. Implementar `atencion_a_clientes`(`in c` : `Cola`[`tuple`[`str`, `int`, `bool`, `bool`]]) $\rightarrow$ `Cola`[`tuple`[`str`, `int`, `bool`, `bool`]] que dada la cola de ingreso de clientes al banco devuelve la cola en la que van a ser atendidos.

3. Diccionarios

En esta sección trabajaremos con el tipo `dict` de Python, que nos permite asociar claves con valores.

Ejercicio 16. Implementar una solución para el siguiente problema.

`problema calcular_promedio_por_estudiante` (in notas: $seq\langle seq\langle \mathsf{Char}\rangle \times \mathbb{R}\rangle$) : Diccionario $\langle$ $seq\langle \mathsf{Char}\rangle$, $\mathbb{R}\rangle$ {
`requiere:` {El primer componente de las tuplas de notas no es una cadena vacía}
`requiere:` {El segundo componente de las tuplas de notas está en el rango [0, 10]}
`asegura:` {Todas las claves de *res* son nombres que aparecen en *notas* (primer componente)}
`asegura:` {Todos los nombres de *notas* (primer componente) son clave en *res*}
`asegura:` {El valor de cada clave de *res* es el promedio de todas las notas que obtuvo el estudiante (primer componente de *notas*)}
}

Ejercicio 17. Se debe desarrollar un navegador web muy simple que debe llevar un registro de los sitios web visitados por los usuarios del sistema. El navegador debe permitir al usuario navegar hacia atrás en la historia de navegación.

1. Crea un diccionario llamado `historiales` que almacenará el historial de navegación para cada usuario. Las claves del diccionario serán los nombres de usuario y los valores serán pilas de String.

2. Implementar una solución para el siguiente problema.

   `problema visitar_sitio` (inout historiales: Diccionario$\langle seq\langle \mathsf{Char}\rangle, Pila[seq\langle \mathsf{Char}\rangle]\rangle$, in usuario: $seq\langle Char\rangle$, in sitio: $seq\langle Char\rangle$)
   {
   `requiere:` {Ninguno de los Strings de los parámetros es vacío}
   `asegura:` {Si *usuario* es una de las claves de *historiales@pre*, entonces se agrega *sitio* a su pila de historiales@pre[usuario]}
   `asegura:` {Si *usuario* no es una de las claves de *historiales@pre*, entonces *historiales*[*usuario*] es igual a la pila que tiene solo el elemento *sitio*}
   `asegura:` {No se modifica ningún otro historial salvo, si existe, el de *usuario*}
   `asegura:` {Todos los pares clave-valor de *historiales@pre* están en *historiales*}
   `asegura:` {Todos los pares clave-valor de *historiales* están en *historiales@pre*, salvo *historiales*[*usuario*] que podría no existir en *historiales@pre*}
   }

3. Implementar una solución para el siguiente problema.

   `problema navegar_atras` (inout historiales: Diccionario$\langle$ $seq\langle Char\rangle$, Pila[$seq\langle Char\rangle$, in usuario: $seq\langle Char\rangle\rangle$) : $seq\langle Char\rangle$
   {
   `requiere:` {Ninguno de los Strings de los parámetros es vacío}
   `requiere:` {*usuario* es una clave de *historiales*}
   `requiere:` {La pila asociada a *usuario* no está vacía}
   `asegura:` {*res* es igual al tope de *historiales@pre*[*usuario*]}
   `asegura:` {*historiales*[*usuario*] es igual a *historiales@pre*[*usuario*] quitando el tope de la pila de *historiales@pre*[*usuario*]}
   `asegura:` {En *historiales*, salvo la pila asociada a *usuario*, no se modifica ningún otro por clave-valor}
   }

   Ejemplo de uso:

```
historiales = {}
visitar_sitio(historiales, "Usuario1", "google.com")
visitar_sitio(historiales, "Usuario1", "facebook.com")
navegar_atras(historiales, "Usuario1")
visitar_sitio(historiales, "Usuario2", "youtube.com")
```

Ejercicio 18. Se debe desarrollar un sistema de gestión de inventario para una tienda de ropa. Este sistema debe permitir llevar un registro de los productos en el inventario y realizar operaciones como agregar nuevos productos, actualizar las existencias y calcular el valor total del inventario.

Para resolver este problema vamos a utilizar un diccionario llamado `inventario` que almacenará la información de los productos. En este diccionario, cada clave será el nombre de un producto, y su valor asociado será otro diccionario con los atributos del producto. Este segundo diccionario tendrá dos claves posibles: 'precio'y 'cantidad', cuyos valores serán de tipo float e int, respectivamente.
Un ejemplo de inventario, con un solo producto, es: {"remera": {"precio": 999.99, "cantidad": 3}}).
Implementar una solución para cada uno de los siguientes problemas. Agregar en las funciones los tipos de datos correspondientes (ver nota al final de la primera especificación).

1. `problema agregar_producto` (inout inventario: Diccionario$\langle$ $seq\langle Char\rangle$, Diccionario$\langle$ $seq\langle Char\rangle$, T $\rangle\rangle$, in nombre: $seq\langle Char\rangle$, in precio: $\mathbb{R}$, in cantidad: $\mathbb{Z}$) {

   `requiere:` {$T \in [\mathbb{Z}, \mathbb{R}]$}

   `requiere:` {$cantidad \geq 0$}

   `requiere:` {$precio \geq 0$}

   `requiere:` {Ninguno de los Strings de los parámetros es vacío}

   `requiere:` {*nombre* no es una clave de *inventario* }

   `asegura:` {Todas los pares clave-valor de *inventario*@*pre* están tal cual en *inventario*}

   `asegura:` {Todas los pares clave-valor de *inventario* están en *inventario*@*pre* y, además, hay una nueva con clave igual a *nombre* y como valor tendrá un diccionario con los pares clave-valor ("precio", *precio*) y ("cantidad", *cantidad*)}

   }

   Se necesitará un diccionario cuyas claves son de tipo String ("precio" y "cantidad") y cuyos valores serán de tipo float y enteros respectivamente. Para declarar los tipos de este diccionario mediante anotaciones en Python, se procede de la siguiente manera:
   En Python 3.9:

   Es necesario importar Union desde el módulo typing para indicar que los valores pueden ser de más de un tipo.
   from typing import Union

   mi_diccionario: dict[str, Union[int, float]]
   En Python 3.10 o superior:

   No es necesario importar Union, ya que se puede usar el operador | para representar una unión de tipos.

   mi_diccionario: dict[str, int | float]

2. `problema actualizar_stock` (inout inventario: Diccionario $\langle$ $seq\langle Char\rangle$, Diccionario$\langle$ $seq\langle Char\rangle$, T $\rangle\rangle$, in nombre: $seq\langle Char\rangle$, in cantidad: $\mathbb{R}$) {

   `requiere:` {$T \in [\mathbb{Z}, \mathbb{R}]$}

   `requiere:` {$cantidad \geq 0$}

   `requiere:` {*nombre* es una clave existente en el *inventario*}

   `requiere:` {Ninguno de los Strings de los parámetros es vacío}

   `asegura:` {Todos los pares clave-valor de *inventario*@*pre* están tal cual en *inventario*, con excepción del que tiene como clave *nombre*}

   `asegura:` {Todos los pares clave-valor de *inventario* están en *inventario*@*pre*}

   `asegura:` {En *inventario*, el valor asociado a la clave *nombre*, tendrá el mismo precio que antes y la cantidad será *cantidad*}

   }

3. `problema actualizar_precio` (inout inventario: Diccionario$\langle$ $seq\langle Char\rangle$, Diccionario$\langle$ $seq\langle Char\rangle$, T $\rangle\rangle$, in nombre:$seq\langle Char\rangle$, in precio: $\mathbb{R}$) {

   `requiere:` {$T \in [\mathbb{Z}, \mathbb{R}]$}

   `requiere:` {$precio \geq 0$}

`requiere:` {*nombre* es una clave existente en el *inventario*}

`requiere:` {Ninguno de los Strings de los parámetros es vacío}

`asegura:` {Todos los pares clave-valor de *inventario@pre* están tal cual en *inventario*, con excepción del valor que tiene como clave *nombre*}

`asegura:` {Todos los pares clave-valor de *inventario* están en *inventario@pre*}

`asegura:` {En *inventario* el diccionario asociado a *nombre*, tendrá la misma cantidad que antes y el precio será *precio*}

}

4. `problema calcular_valor_inventario` (in inventario: Diccionario $\langle$ $seq\langle Char\rangle$, Diccionario $\langle$ $seq\langle Char\rangle$, T $\rangle\rangle$) : $\mathbb{R}$ {

   `requiere:` {$T \in [\mathbb{Z}, \mathbb{R}]$}

   `requiere:` {Ninguno de los Strings del inventario es vacío}

   `asegura:` {*res* es la suma, para cada producto, del precio multiplicado por la cantidad}

   }

Ejemplo de uso:

```
inventario = {}
agregar_producto(inventario, "Camisa", 20.0, 50)
agregar_producto(inventario, "Pantalón", 30.0, 30)
actualizar_stock(inventario, "Camisa", 10)
valor_total = calcular_valor_inventario(inventario)
print("Valor total del inventario:", valor_total) # Debería imprimir 1100.0
```

4. Archivos

Para usar archivos contamos con las funciones: open, close, read, readline, readlines, write, os.path.join(), os.path.exists(). Para más información referirse a la documentación: `https://docs.python.org/es/3/tutorial/inputoutput.html#reading-and-writing-files`

Ejercicio 19. Implementar una solución para cada uno de los siguientes problemas.

1. `problema contar_lineas` (in nombre_archivo: $seq\langle Char\rangle$) : $\mathbb{Z}$ {

   `requiere:` {*nombre_archivo* es el path con el nombre de un archivo existente y accesible}

   `asegura:` {*res* es igual a la cantidad de líneas que contiene el archivo indicado por *nombre_archivo*}

   }

2. `problema existe_palabra` (in nombre_archivo: $seq\langle Char\rangle$, in palabra: $seq\langle Char\rangle$) : Bool {

   `requiere:` {*nombre_archivo* es el path con el nombre de un archivo existente y accesible}

   `requiere:` {*palabra* no es vacía}

   `asegura:` {*res* es verdadero si y solo si *palabra* aparece al menos una vez en el archivo indicado por *nombre_archivo*}

   }

3. `problema cantidad_de_apariciones` (in nombre_archivo: $seq\langle Char\rangle$, in palabra: $seq\langle Char\rangle$) : $\mathbb{Z}$ {

   `requiere:` {*nombre_archivo* es el path con el nombre de un archivo existente y accesible}

   `requiere:` {*palabra* no es vacía}

   `asegura:` {*res* es la cantidad de veces que *palabra* aparece en el archivo indicado por *nombre_archivo*}

   }

Ejercicio 20. Implementar una solución para el siguiente problema.

`problema agrupar_por_longitud` (in nombre_archivo: $seq\langle Char\rangle$) : Diccionario$\langle\mathbb{Z},\mathbb{Z}\rangle$ {
`requiere:` {*nombre_archivo* es el path con el nombre de un archivo existente y accesible}
`asegura:` {Para cada longitud n tal que existe al menos una palabra de longitud n en el archivo indicado por *nombre_archivo*, $res[n]$ es igual a la cantidad de palabras de esa longitud}
`asegura:` {No hay otras claves en *res* que no correspondan a longitudes de palabras presentes en el archivo}
}

Por ejemplo, el diccionario

```
{
        1: 2,
        2: 10,
        5: 4
}
```

indica que se encontraron 2 palabras de longitud 1, 10 palabras de longitud 2 y 4 palabras de longitud 5. Para este ejercicio se consideran como palabras todas aquellas secuencias de caracteres delimitadas por espacios en blanco.

Ejercicio 21. Implementar una solución para el siguiente problema.

`problema la_palabra_mas_frecuente` (in nombre_archivo: $seq\langle Char\rangle$) : $seq\langle Char\rangle$ {
`requiere:` {*nombre_archivo* es un archivo existente y accesible que tiene, por lo menos, una palabra}
`asegura:` {*res* es una palabra que aparece en el archivo *nombre_archivo*}
`asegura:` {No hay ninguna palabra contenida en el archivo *nombre_archivo* que aparezca más veces que la palabra *res* }
}

Para resolver el problema se aconseja utilizar un diccionario de palabras.

Ejercicio 22. Implementar una solución para el siguiente problema.

`problema clonar_sin_comentarios` (in nombre_archivo_entrada: $seq\langle Char\rangle$, in nombre_archivo_salida: $seq\langle Char\rangle$) {
`requiere:` {*nombre_archivo_entrada* es el path con el nombre de un archivo existente y accesible}
`requiere:` {*nombre_archivo_salida* es el path con el nombre de un archivo que, si existe, se puede modificar, y si no existe, se puede crear}
`asegura:` {El archivo indicado por *nombre_archivo_salida* contiene las mismas líneas y en el mismo orden que el archivo *nombre_archivo_entrada*, excepto aquellas que comienzan con el carácter `#`}
}

Para este ejercicio vamos a considerar que una línea es un comentario si tiene un '#'como primer carácter de la línea, o si no es el primer carácter, se cumple que todos los anteriores son espacios.

Por ejemplo, si se llama a *clonar_sin_comentarios* con un archivo con este contenido:

```
# esto es un comentario
# esto tambien
esto no es un comentario # esto tampoco
```

nombre_archivo_salida solo contendrá la última línea:

```
esto no es un comentario # esto tampoco
```

Ejercicio 23. Implementar una solución para el siguiente problema.

`problema invertir_lineas` (in nombre_archivo_entrada: $seq\langle Char\rangle$, in nombre_archivo_salida: $seq\langle Char\rangle$) {
`requiere:` {*nombre_archivo_entrada* es el path de un archivo de texto existente y accesible}
`requiere:` {*nombre_archivo_salida* es el path con el nombre de un archivo que, si existe, se puede modificar, y si no existe, se puede crear}
`asegura:` {El archivo indicado por *nombre_archivo_salida* contiene las mismas líneas que el archivo *nombre_archivo_entrada*, pero en orden inverso}
}

Por ejemplo, si el archivo contiene lo siguiente:

```
Esta es la primera linea.
Y esta es la segunda.
```

debe generar:

```
Y esta es la segunda.
Esta es la primera linea.
```

Ejercicio 24. Implementar una solución para el siguiente problema.

`problema agregar_frase_al_final` (in nombre_archivo: $seq\langle Char\rangle$, in frase: $seq\langle Char\rangle$) {
    `requiere:` {*nombre_archivo* es el path de un archivo existente y accesible}
    `requiere:` {*frase* no es vacía}
    `asegura:` {*frase* se agrega como una nueva línea al final del archivo *nombre_archivo*}
}

Este problema no crea una copia del archivo de entrada, sino que lo modifica.

Ejercicio 25. Dado un archivo de texto y una frase, implementar una función `agregar_frase_al_principio`(`in nombre_archivo` : `str`, `in frase` : `str`), que agregue la frase al comienzo del archivo original (similar al ejercicio anterior, sin hacer una copia del archivo).

`problema agregar_frase_al_principio` (in nombre_archivo: $seq\langle Char\rangle$, in frase: $seq\langle Char\rangle$) {
    `requiere:` {*nombre_archivo* es el path de un archivo existente y accesible}
    `requiere:` {*frase* no es vacía}
    `asegura:` {*frase* se agrega como primera línea del archivo *nombre_archivo*, desplazando las anteriores hacia abajo}
}

Este problema no crea una copia del archivo de entrada, sino que lo modifica.

Ejercicio 26. Implementar una solución para el siguiente problema.

`problema listar_palabras_de_archivo` (in nombre_archivo: $seq\langle Char\rangle$) : $seq\langle seq\langle Char\rangle\rangle$ {
    `requiere:` {*nombre_archivo* es el path de un archivo existente y accesible}
    `asegura:` {*res* contiene exactamente las palabras legibles distintas que aparecen en el archivo *nombre_archivo*}
}

Definimos una palabra legible como:

- secuencias de texto formadas por números, letras mayúsculas/minúsculas y los caracteres ‘ ’(espacio) y ‘_’(guion bajo)
- que tienen longitud >= 5

Referencia: https://docs.python.org/es/3/library/functions.html#open

> Para resolver este ejercicio se puede abrir un archivo en modo binario `'b'`. `Al hacer read() vamos a obtener una secuencia de bytes, que al hacer chr(byte) nos va a devolver un carácter correspondiente al byte leído.`

Una vez implementada la función, probarla con diferentes archivos binarios (.exe, .zip, .wav, .mp3, etc).

Ejercicio 27. Implementar una solución para el siguiente problema.

`problema calcular_promedio_por_estudiante` (in nombre_archivo_notas: $seq\langle Char\rangle$, in nombre_archivo_promedios: $seq\langle Char\rangle$)
{
    `requiere:` {*nombre_archivo_notas* es el path de un archivo existente y accesible, con formato CSV: cada línea tendrá número de LU, materia, fecha y nota, todo separado por comas}
    `requiere:` {*nombre_archivo_promedios* es el path de un archivo distinto, accesible para escritura}
    `asegura:` {El archivo *nombre_archivo_promedios* contiene una línea por estudiante del archivo *nombre_archivo_notas*, con su LU y su promedio separados por una coma}
}

El contenido del archivo *nombre_archivo_notas* tiene el siguiente formato:

`nro de LU (str), materia (str), fecha (str), nota (float)`

Analizar el problema y modularizar el código apropiadamente. Una opción es implementar una función auxiliar que cumpla la siguiente especificación.

`problema promedio_estudiante` (in notas_de_estudiantes: $seq\langle seq\langle Char\rangle\rangle$, in lu: $seq\langle Char\rangle$) : $\mathbb{R}$ {

`requiere:` {*notas_de_estudiantes* tiene el contenido del archivo de notas. Cada elemento de la lista es una línea de ese archivo, con formato CSV: tendrá número de LU, materia, fecha y nota, todo separado por comas}

`requiere:` {*lu* corresponde a una LU presente en *notas_de_estudiantes*}

`asegura:` {*res* es el promedio de las notas asociadas a *lu* en *notas_de_estudiantes*}

}