Ficha 9

Procesamiento de Secuencias de Caracteres

1.] Procesamiento de secuencias de caracteres.

Esta sección (y de hecho, esta Ficha completa) está orientada al planteo de una situación clásica como es el procesamiento de una secuencia de caracteres que forman una frase, de forma tal que el programador no sólo debe ser capaz de identificar palabras dentro de esa frase, sino también la formación de ciertos patrones en cada palabra. Esta clase de problemas forma parte de un excelente campo para el desarrollo de habilidades algorítmicas que ya dejan de ser triviales para un programador que recién se inicia [2]. El enunciado que se sugiere para comenzar el análisis es el siguiente:

Problema 24.) *Desarrollar un programa en Python que permita cargar por teclado **un texto completo** (analizar dos opciones: una es cargar todo el texto en una variable de tipo cadena de caracteres y recorrerla con un for iterador; y la otra es cargar cada caracter uno por uno a través de un while). Siempre se supone que el usuario cargará **un punto** para indicar el final del texto, y que cada palabra de ese texto está separada de las demás por **un espacio en blanco**. El programa debe:*

a. Determinar cuántas palabras se cargaron.
b. Determinar cuántas palabras comenzaron con la letra "p".
c. Determinar cuántas palabras contuvieron una o más veces la expresión "ta".

Discusión y solución: Este tipo de problema es especialmente desafiante porque implica numerosos puntos de vista para resolverlo, además de muchas variantes en cuanto al planteo posible de la carga y el soporte de datos. Como veremos, es también muy valioso en cuanto a que permite aplicar y dominar técnicas básicas como el uso profundo de banderas o flags, carga por doble lectura, ajuste de un ciclo para forzarlo a ser *[1..N]*, uso de ciclos iteradores, condiciones anidadas, y cuanta idea pudiera ser útil para llegar a una solución.

Por lo pronto, podemos suponer que en nuestra *primera versión* el texto será ingresado en forma "inocente": cada una de las "letras" ingresará de a una por vez, una por cada vuelta del ciclo de control. Como sabemos que el final del texto se indica con la carga de un punto, entonces inicialmente ese ciclo puede ser un *while con doble lectura*, en base al esquema general que sigue[1]:

[1] La idea de la llegada de alguna forma de mensaje (de texto o del tipo que sea) para ser procesado e interpretado ha conducido a pensar en qué pasaría si se recibiese un mensaje o comunicación desde alguna avanzada civilización extraterrestre. A lo largo de los años se han creado programas de investigación (como el *Programa SETI: Search for ExtraTerrestrial Intelligence*) orientados a captar e interpretar alguno de estos eventuales mensajes. Y el cine se hizo cargo del tema al menos en una conocida película de 1997: *Contact* (conocida como *Contacto* en español), dirigida por *Robert Zemeckis* e interpretada por *Jodie Foster*. La película es la adaptación cinematográfica de la novela del mismo nombre escrita por el famosísimo científico *Carl Sagan*, y especula sobre la reacción de la civilización humana frente a la llegada de un mensaje de una civilización de otro planeta.

```
car = input('Letra (con "." termina):  ')
while car != '.':
    # procesar el caracter ingresado en car

    # cargar el siguiente caracter
    car = input('Letra (con "." termina):  ')
```

Sin embargo, este planteo puede traer luego algún tipo de incomodidad al tener que controlar posibles nuevos caracteres *que se cargan en dos lugares diferentes* del proceso. Por ese motivo, y para simplificar a futuro, podemos también convertir el *while de doble lectura* en un *while forzado a trabajar como ciclo [1..N]* (que podemos referir de ahora en más como *while-[1..N]*), y hacer una única lectura al inicio del bloque de acciones del ciclo:

```
car = None
while car != '.':
    car = input('Letra (con "." termina):  ')
    # procesar el caracter ingresado en car
```

La inicialización de *car* con el valor *None* hace que *car* quede definida (y por lo tanto exista) *antes* de chequear si su valor es diferente de un punto. Y como el valor *None* es efectivamente diferente de un punto, la condición de control del ciclo será indefectiblemente verdadera en el primer chequeo, haciendo que la ejecución de la primera vuelta del ciclo esté garantizada.

Lo siguiente es controlar si el caracter que se acaba de ingresar en la variable *car* es concretamente una letra (en cuyo caso hay que procesarlo como parte de una palabra), o bien determinar si ese caracter es un espacio en blanco (que no debe ser procesado como una letra sino como un terminador de palabra). En principio, el control es simple:

```
car = None
while car != '.':
    car = input('Letra (con "." termina):  ')
    # procesar el caracter ingresado en car
    # ...

    # final de palabra?
    if car == ' ':
        # ha terminado una palabra...
    else:
        # car es una letra... la palabra sigue...
```

Lo anterior permite detectar si el caracter que se acaba de cargar en *car* es un espacio en blanco y actuar en consecuencia con la palabra que acaba de terminar. Sin embargo, revisando con cuidado el enunciado podemos darnos cuenta que si bien en general todas las palabras finalizan con un blanco, hay exactamente una que finaliza con otro caracter: la *última palabra del texto, que termina con un punto* (el mismo punto que también da por terminado al texto completo). Ese caso particular debe ser previsto, y es simple de hacer:

```
car = None
while car != '.':
    car = input('Letra (con "." termina):  ')
    # procesar el caracter ingresado en car
    # ...

    # final de palabra?
    if car == ' ' or car == '.':
        # ha terminado una palabra...
    else:
        # car es una letra... la palabra sigue...
```

Si el caracter cargado en *car* es un blanco o es un punto, la palabra que se estaba procesando ha finalizado, y pueden aplicarse los procesos que sean requeridos en ese caso (sobre los cuales volveremos luego).

Teniendo definido el *esquema de control de fin de palabra*, tenemos que analizar ahora la forma de procesar cada letra para cumplir con los requerimientos del enunciado. El primero (*determinar cuántas palabras se cargaron*) es simple. Necesitamos un contador (que llamaremos *ctp* por *contador total de palabras*) inicializado en cero antes del ciclo de carga, y de forma que sume uno cada vez que se detecta que una palabra ha terminado. Cuando el ciclo se detenga, sencillamente se muestra el valor de *ctp*:

```
ctp = 0
car = None
while car != '.':
    car = input('Letra (con "." termina):  ')
    # procesar el caracter ingresado en car
    # ...

    # final de palabra?
    if car == ' ' or car == '.':
        # ha terminado una palabra...

        # ...contarla...
        ctp += 1
    else:
        # car es una letra... la palabra sigue...
print('Cantidad de palabras:', ctp)
```

Lo anterior sería estrictamente suficiente para resolver el caso. Sin embargo, un buen programador haría bien es desconfiar y buscar algún punto débil en el planteo. Y en este caso existe uno: el programa está planteado para el supuesto de que el usuario cargará palabras separadas por un blanco y terminando la última con un punto, y mantendremos esa suposición para hacer simple el planteo. Pero específicamente, podría ocurrir que el usuario cargue un blanco (o un punto) *directamente en la primera lectura* (en la primera vuelta del ciclo). Y si ese fuese el caso, el programa contaría ese blanco (o ese punto) como una palabra, lo cual es claramente incorrecto (asegúrese de entender este hecho haciendo una prueba de escritorio rápida antes de continuar leyendo el resto de la explicación…)

Para eliminar ese molesto caso, se puede incorporar al programa un contador de letras (que llamaremos ***cl***). La idea es que cada vez que se cargue un caracter se incremente ese contador. Al terminar una palabra, chequear el valor de ***cl*** y contar la palabra en *ctp* sólo si ***cl*** es mayor a 1 (si ***cl*** es 1, el caracter contado fue el blanco o el punto, y si vale más de 1 es porque necesariamente se cargó algún otro caracter adicional):

```
# contador total de palabras
ctp = 0

# contador de letras en una palabra
cl = 0

car = None
while car != '.':
    # cargar y procesar el caracter ingresado en car
    car = input('Letra (con "." termina):  ')
    cl += 1

    # final de palabra?
```

```
        if car == ' ' or car == '.':
            # ha terminado una palabra...

            # ...contarla solo si hubo al menos una letra...
            if cl > 1:
                ctp += 1

            # reiniciar contador de letras (por próxima palabra)...
            cl = 0
        else:
            # car es una letra... la palabra sigue...
    print('Cantidad de palabras:', ctp)
```

En el esquema anterior, una vez que la palabra ha sido contada en *ctp*, el contador de letras ***cl*** debe volver al valor 0 antes de comenzar a procesar la palabra que sigue, pues de otro modo seguirá contando letras en forma acumulativa: en lugar de contar las letras de una palabra, estaría contando todas las letras del texto.

El segundo requerimiento es *determinar cuántas palabras comenzaron con la letra "p"*, lo cual es un poco más complejo. Está claro que ahora necesitamos saber si la letra cargada es o no es una "p", pero además queremos saber si esa "p" es la primera letra de la palabra actual. En este caso la solución es directa ya que contamos con el contador de letras ***cl*** que hemos usado en el caso anterior: si la letra es una "p" y el contador de letras vale 1, eso significa que efectivamente la primera letra de la palabra es una "p" y podemos entonces contar esa palabra con otro contador (que llamaremos ***cpp***, por *cantidad de palabras con p*):

```
# contador total de palabras
ctp = 0

# contador de letras en una palabra
cl = 0

# contador de palabras que empiezan con p
cpp = 0

car = None
while car != '.':
    # cargar y procesar el caracter ingresado en car
    car = input('Letra (con "." termina):  ')
    cl += 1

    # final de palabra?
    if car == ' ' or car == '.':
        # ha terminado una palabra...

        # ...contarla solo si hubo al menos una letra...
        if cl > 1:
            ctp += 1

        # ...reiniciar contador de letras (por próxima palabra)...
        cl = 0
    else:
        # car es una letra... la palabra sigue...
        # ...contar la palabra si comienza con "p"...
        if cl == 1 and car == 'p':
            cpp += 1

print('Cantidad de palabras:', ctp)
print('Cantidad de palabras que empiezan con "p":', cpp)
```

De nuevo, al finalizar la carga (por la aparición del punto) se muestra en pantalla el valor final del contador ***cpp***, y el segundo requerimiento queda así cumplido.

El tercer y último requerimiento es *determinar cuántas palabras contenían al menos una vez la expresión (o sílaba) "ta"*, para lo que habrá que trabajar un poco más. Veamos: si el caracter es una letra, nos interesa por el momento saber si es una "t" y dejar marcado de alguna forma ese hecho para que al ingresar el siguiente caracter podamos comprobar si el mismo es una "a" y el anterior una "t". Hay muchas formas de hacer esto, pero puede resolverse con *flags* o *banderas* para marcar los estados que nos interesan.

Por lo pronto, usaremos una bandera llamada ***st*** (por *señal de la letra t*) para avisar si la última letra cargada fue efectivamente una "t" (con *st = True*) o no (*st = False*):

```
# contador total de palabras
ctp = 0

# contador de letras en una palabra
cl = 0

# contador de palabras que empiezan con p
cpp = 0

# flag: la ultima letra fue una "t"?
st = False

car = None
while car != '.':
    # cargar y procesar el caracter ingresado en car
    car = input('Letra (con "." termina):  ')
    cl += 1

    # final de palabra?
    if car == ' ' or car == '.':
        # ha terminado una palabra...
        # ...contarla solo si hubo al menos una letra...
        if cl > 1:
            ctp += 1

        # ...reiniciar contador de letras (por próxima palabra)...
        cl = 0
    else:
        # car es una letra... la palabra sigue...
        # ...contar la palabra si comienza con "p"...
        if cl == 1 and car == 'p':
            cpp += 1

        # ...deteccion de la silaba "ta"...
        # ...por ahora, solo avisar si la ultima fue una "t"...
        if car == 't':
            st = True
        else:
            st = False

print('Cantidad de palabras:', ctp)
print('Cantidad de palabras que empiezan con "p":', cpp)
```

El primer paso (mostrado en el esquema anterior) "reacciona" al paso de una letra "t" cambiando el estado del flag ***st*** a *True* o *False* de acuerdo a si el valor cargado en *car* en ese momento es o no una "t".

El segundo paso es ajustar ese esquema para que ahora reaccione al posible paso de una "a" ***inmediatamente luego*** del paso de una "t". Eso puede hacerse con otro flag que llamaremos ***sta*** (por *señal de la sílaba "ta"*) y usando un poco de ingenio:

```
# contador total de palabras
ctp = 0

# contador de letras en una palabra
cl = 0

# contador de palabras que empiezan con p
cpp = 0

# contador de palabras que tuvieron "ta"
cta = 0

# flag: la ultima letra fue una "t"?
st = False

# flag: se ha formado la silaba "ta" al menos una vez?
sta = False

car = None
while car != '.':
    # cargar y procesar el caracter ingresado en car
    car = input('Letra (con "." termina):  ')
    cl += 1

    # final de palabra?
    if car == ' ' or car == '.':
        # ha terminado una palabra...
        # ...contarla solo si hubo al menos una letra...
        if cl > 1:
            ctp += 1

        # si hubo "ta" contar la palabra...
        if sta == True:
            cta += 1

        # ...reiniciar contador de letras (por próxima palabra)...
        cl = 0
        # reiniciar flags (por próxima palabra)...
        st = sta = False
    else:
        # car es una letra... la palabra sigue...
        # ...contar la palabra si comienza con "p"...
        if cl == 1 and car == 'p':
            cpp += 1

        # ...deteccion de la silaba "ta"...
        if car == 't':
            st = True
        else:
            # hay una "a" y la anterior fue una "t"?...
            if car == 'a' and st == True:
                sta = True
            st = False

print('Cantidad de palabras:', ctp)
print('Cantidad de palabras que empiezan con "p":', cpp)
print('Cantidad de palabras que contienen "ta":', cta)
```

El esquema anterior resuelve el problema. Toda palabra que efectivamente contenga la sílaba "ta" será detectada marcando en *True* la bandera ***sta***, y contando esa palabra en el contador ***cta*** cuando la palabra termine.

Asegúrese de entender cómo funciona este "*detector de la sílaba ta*" haciendo pruebas de escritorio detalladas con palabras como "tano" (deja ***sta*** en *True*), "tea" (deja ***sta*** en *False*), "patata" (deja ***sta*** en *True* y *obviamente la palabra es contada sólo una vez*) o "pala" (deja ***sta*** en *False*). El programa completo puede plantearse así (dispone del fuente completo en el archivo *letritas01.py* de la carpeta *Fuentes*, que acompaña a esta ficha).

```
# titulo general...
print('Procesamiento de una secuencia de caracteres')
print('Version 1: cargando los caracteres uno a uno...')

# inicializacion de flags y contadores...
# contador de letras en una palabra...
cl = 0

# contador total de palabras...
ctp = 0

# contador de palabras que empiezan con "p"...
cpp = 0

# contador de palabras que tienen la expresion "ta"...
cta = 0

# flag: la ultima letra vista fue una "t"?...
st = False

# flag: se ha formado la silaba "ta" al menos una vez?...
sta = False

# instrucciones en pantalla...
print('Ingrese las letras del texto, pulsando <Enter> una a una')
print('(Para finalizar la carga, ingrese un punto)')
print()

# ciclo de carga - [1..N] (primera vualta forzada)...
car = None
while car != '.':
    # cargar proxima letra y contarla...
    car = input('Letra (con punto termina): ')
    cl += 1

    # fin de palabra?
    if car == ' ' or car == '.':
        # 1.) contar la palabra solo si hubo al menos una letra...
        if cl > 1:
            ctp += 1

        # 2.) si hubo 'ta' contar la palabra...
        if sta:
            cta += 1

        # reiniciar contador de letras...
        cl = 0

        # reiniciar flags...
        st = sta = False
```

```
            # regresar al ciclo ahora mismo...
            continue

        # 2.) deteccion de palabras que empiezan con "p"...
        if cl == 1 and car == 'p':
            cpp += 1

        # 3.) deteccion de expresion "ta"...
        if car == 't':
            st = True
        else:
            if car == 'a' and st:
                sta = True
            st = False

# visualización de los resultados finales...
print('1. Cantidad total de palabras:', ctp)
print('2. Cantidad de palabras que empiezan con "p":', cpp)
print('3. Cantidad de palabras con la expresion "ta":', cta)
```

Sólo queda por aclarar un detalle técnico menor: dentro del ciclo de carga, la rama verdadera de la condición que detecta un blanco o un punto para saber si ha terminado una palabra, contiene una instrucción *continue* al final:

```
            # fin de palabra?
            if car == ' ' or car == '.':
                # 1.) contar la palabra si hubo al menos una letra...
                if cl > 1:
                    ctp += 1

                # 2.) si hubo 'ta' contar la palabra...
                if sta:
                    cta += 1

                # reiniciar contador de letras...
                cl = 0

                # reiniciar flags...
                st = sta = False

                # regresar al ciclo ahora mismo...
                continue
```

Esta instrucción (como se vio en una ficha anterior) hace que se regrese inmediatamente a la cabecera del ciclo dentro del cual está incluida, sin ejecutar ninguna de las instrucciones que estuviesen escritas debajo de ella. En este caso, el uso de *continue* al terminar la rama verdadera hace que ya no sea necesario el *else* para esa condición, simplificando mínimamente la estructura del código fuente (al no tener que escribir *else* ni tener que indentar toda esa rama).

Si bien el programa anterior resuelve por completo el problema, subsiste un detalle referido a la interfaz de usuario que es bastante molesto e incómodo: los caracteres deben cargarse uno por uno y presionar <Enter> con cada uno… incluidos los espacios en blanco y el punto final. La ejecución del programa para cargar sólo la frase "la tea." produce una sesión de carga como la siguiente:

```
Deteccion de palabras con la expresion "ta"
Version 1: cargando los caracteres uno a uno...
```

```
Ingrese las letras del texto, pulsando <Enter> una a una
(Para finalizar la carga, ingrese un punto)

Letra (con punto termina): l
Letra (con punto termina): a
Letra (con punto termina):
Letra (con punto termina): t
Letra (con punto termina): e
Letra (con punto termina): a
Letra (con punto termina): .

1. Cantidad total de palabras: 2
2. Cantidad de palabras que empiezan con "p": 0
3. Cantidad de palabras con la expresion "ta": 0
```

Y se puede ver que esto se complica aún más si el texto a procesar fuese mucho más largo o bien si el programa debiese ser ejecutado muchas veces para hacer pruebas. Por ese motivo podemos sugerir que el texto se cargue "todo junto" en una sola lectura por teclado, almacenándolo directamente en una variable de tipo cadena de caracteres. Una vez cargada esa cadena (que también ***debe*** finalizar con un punto para que sea correctamente detectada la última palabra), se puede procesar la secuencia mediante un *for iterador*, respetando el espíritu del enunciado en cuanto a analizar uno a uno los caracteres. La conversión del programa para adaptarlo a estas ideas, podría verse como sigue (dispone del fuente completo en el archivo *letritas02.py* de la carpeta *Fuentes*, que acompaña a esta ficha).):

```
# titulo general...
print('Deteccion de palabras con la expresion "ta"')
print('Version 2: cargando todo el texto en una cadena...')

# inicializacion de flags y contadores...
# contador de letras en una palabra...
cl = 0

# contador total de palabras...
ctp = 0

# contador de palabras que empiezan con "p"...
cpp = 0

# contador de palabras que tienen la expresion "ta"...
cta = 0

# flag: la ultima letra vista fue una "t"?...
st = False

# flag: se ha formado la silaba "ta" al menos una vez?...
sta = False

# carga del texto completo...
cadena = input('Cargue el texto completo (finalizando con un punto): ')

# ciclo iterador para procesar el texto...
for car in cadena:
    # contar la letra actual...
    cl += 1

    # fin de palabra?
    if car == ' ' or car == '.':
        # 1.) contar la palabra solo si hubo al menos una letra...
```

```
            if cl > 1:
                ctp += 1

            # 2.) si hubo 'ta' contar la palabra...
            if sta:
                cta += 1

            # reiniciar contador de letras...
            cl = 0

            # reiniciar flags...
            st = sta = False

            # regresar al ciclo ahora mismo...
            continue

        # 2.) deteccion de palabras que empiezan con "p"...
        if cl == 1 and car == 'p':
            cpp += 1

        # 3.) deteccion de expresion "ta"...
        if car == 't':
            st = True
        else:
            if car == 'a' and st:
                sta = True
            st = False

# visualización de los resultados finales...
print('1. Cantidad total de palabras:', ctp)
print('2. Cantidad de palabras que empiezan con "p":', cpp)
print('3. Cantidad de palabras con la expresion "ta":', cta)
```

Como se puede ver, el esquema lógico es exactamente el mismo. El uso de una cadena en lugar de la lectura "caracter a caracter" hace más sencillo el manejo del programa al ejecutarlo, y por lo tanto también se simplifica el proceso de prueba. Pero por lo demás, desde el punto de vista lógico, es tan válida una técnica como la otra.

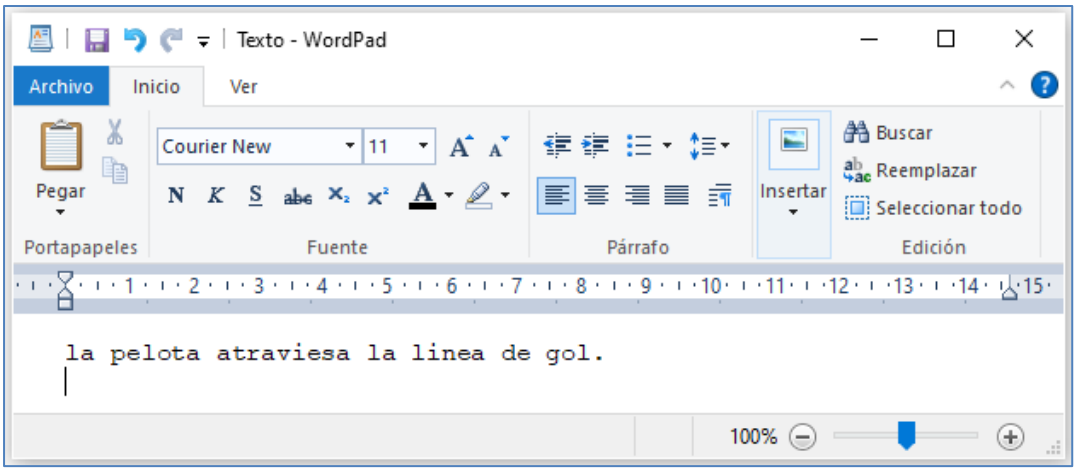
2.] Carga de la secuencia a procesar desde un archivo de texto.

La carga por teclado de la cadena o secuencia de caracteres a procesar en un programa como el que se expuso en la sección anterior puede hacerse sin problemas, pero muchas veces el programador querrá evitar esa carga manual y tomar la cadena desde un *archivo de texto simple*. Esto puede ser muy útil por ejemplo, en casos de evaluaciones en las que se pretende que todos los estudiantes carguen y procesen exactamente la misma secuencia, o en contextos en los que se está desarrollando el programa y se requieren numerosas pruebas de funcionamiento antes de darlo por terminado (sin tener que cargar en cada prueba a mano la secuencia de entrada).

El procesamiento de archivos de texto será estudiado en profundidad más adelante, en una ficha posterior. Pero los detalles básicos para poder abrir un archivo de texto simple, leer su contenido, asignar ese contenido en una variable (que será obviamente de tipo cadena) y luego cerrar el archivo para finalmente procesar la cadena (en un programa como el de la sección anterior, por caso), es simple de introducir y es justamente lo que haremos en esta sección complementaria.

Por lo pronto, asumiremos que el archivo que queremos leer contiene texto con el formato indicado para la entrada de un programa como el del final de la sección anterior: el texto consiste en una secuencia de palabras separadas estrictamente con un blanco (y solo un blanco), y de forma tal que toda la secuencia finaliza con un punto. Un archivo como ese puede ser creado obviamente por un programa Python (como veremos en una ficha posterior), pero también con cualquier editor de textos (por ejemplo, *NotePad*, *WordPad* o cualquier otro que pueda grabar en formato de texto plano [archivos con extensión *.txt*]). La *Figura 1* siguiente muestra un ejemplo de un archivo de texto llamado *Texto.txt* conteniendo una simple línea de texto con este formato, creado y abierto con *WordPad*:

Figura 1: Ejemplo de un archivo de texto simple creado y desplegado con *WordPad*.



Para procesar el contenido de un archivo de texto (o de cualquier otro tipo) ese archivo debe ser *abierto* (o sea, se debe habilitar el canal de comunicación entre el dispositivo externo que contiene a ese archivo y la memoria principal donde está el programa). En Python la apertura de un archivo se hace mediante la función predefinida *open()*, en forma similar lo que sigue:

```
m = open(nombre_del archivo, modo_de_apertura)
```

En el modelo anterior, nombre_del_archivo es una cadena de caracteres que indica el nombre del archivo que se quiere abrir, y modo_de_apertura es otra cadena de caracteres que indica el tipo del archivo y el modo en que se quiere abrir a ese archivo. Si nuestra intención fuese abrir el archivo de texto llamado Texto.txt de forma de poder leer su contenido (sin necesidad de grabar), entonces la instrucción sería la que sigue:

```
m = open('Texto.txt', 'rt')
```

El nombre del archivo puede contener la ruta del directorio o carpeta que lo contiene (si esa ruta no se indica, Python asume que el archivo está en la misma carpeta del proyecto):

```
m = open('c:\\documentos\\Texto.txt', 'rt')
```

El modo de apertura es una cadena simple de dos o tres caracteres. El último caracter será una *"t"* para indicar que el archivo es de texto, o una *"b"* para indicar que el archivo es un archivo binario (no necesariamente representando texto en su interior). Este segundo caracter puede obviarse, y en ese caso Python asumirá que el archivo es de texto. La primera instrucción que vimos más arriba, sería entonces equivalente a:

```
m = open('Texto.txt', 'r')
```

El primer caracter del modo de apertura será una *"r"* para indicar que el archivo se abre en modo de solo lectura (solo se permitirán lecturas en ese archivo), una *"w"* para indicar que se abre en modo de solo escritura (solo se permitirán grabaciones en ese archivo), o una *"a"* para indicar que se abre en modo de solo escritura, pero de forma que cualquier grabación se hará al final del archivo (a diferencia del modo *"w"* permite grabar en cualquier lugar dentro del archivo). Si el modo de apertura se omite completamente, Python asumirá que el archivo es de texto, y que será abierto en modo *"r"* de solo lectura. La última instrucción que vimos es equivalente entonces a:

```
m = open('Texto.txt')
```

Un detalle importante a considerar, es que si el modo de apertura es "r", entonces el archivo que se quiere abrir ***debe*** existir previamente en la ruta indicada o asumida por Python. Si el archivo no existe, y se intenta abrir en modo *"r"*, la función *open()* lanzará un error y el programa se interrumpirá.

Otros modos de apertura serán analizados en una ficha posterior (por ahora es suficiente con los que se han indicado).

La función *open()* abre el archivo en el modo pedido, y retorna un objeto que es asignado en la variable de la izquierda del operador de asignación. En todos nuestros ejemplos anteriores es la variable *m*. A todos los efectos prácticos, esa variable, de ahí en más, representa al archivo que se abrió, y a través de ella se aplican las operaciones para procesar el archivo.

Una vez abierto el archivo de texto, solo resta saber que se puede leer todo el contenido del archivo con el método *read()* (veremos en fichas posteriores la diferencia entre métodos y funciones) que se accede desde la variable que representa al archivo.

```
m = open('Texto.txt', 'rt')
cad = m.read()
```

Note que este método *lee el archivo completo* y retorna una copia de ese contenido. Esa copia normalmente se asigna en una variable, que obviamente será de tipo *str* (cadena de caracteres. En el ejemplo anterior la copia fue almacenada en la variable *cad*. Y básicamente, eso es todo… El texto que el archivo contenía, está desde ese momento copiado en la variable *cad*, y el programador puede procesar esa cadena en la forma que necesite (sin tener que cargarla por teclado…).

Lo único que resta por saber es un detalle técnico: cuando un archivo termina de procesarse, debería ser *cerrado*. Si bien el sistema operativo cierra automáticamente cualquier archivo abierto que haya quedado luego de ejecutar un programa, es buena idea que el programador se acostumbre a cerrar cualquier archivo que haya abierto para evitar eventual pérdida de datos (la razón por la que puede producirse pérdida de datos será analizada en una ficha posterior, pero vale aclarar que esa eventual pérdida puede producirse solo si el archivo se abre para ser grabado).

El cierre de un archivo, se hace con el método *close()*, y ese método simplemente cierra el canal de comunicación que *open()* abrió para gestionar el archivo entre el dispositivo y la memoria. La variable que representaba al archivo (*m* en nuestro caso) a partir de ese momento queda indefinida (no puede seguir usándose para manejar el archivo, a menos que se use open() nuevamente para volver a abrirlo). El script que sigue muestra el proceso completo:

```
m = open('Texto.txt', 'rt')
```

```
cad = m.read()
m.close()
# procesar aquí la cadena cad...
```

Como puede ver, si queremos procesar una cadena al estilo de lo que mostramos en la sección anterior, pero tomando esa cadena desde un archivo de texto y no desde el teclado, solo hay reemplazar la instrucción *input()* que hacía la carga por teclado, por tres instrucciones como las anteriores. Mostramos a modo de ejemplo el mismo programa que desarrollamos para resolver el problema 24 de la sección anterior, pero aplicando estas ideas ahora. Solo asegúrese de tener el archivo *Texto.txt* ya creado antes de ejecutar este programa, y de tenerlo almacenado en la misma carpeta donde esté el programa. Por supuesto, para que el programa funcione correctamente, se espera que el texto en ese archivo contenga palabras separadas por un único espacio en blanco, y que todo el texto finalice con un punto. La longitud de la cadena no tiene importancia: el programa hará su trabajo correctamente (dispone del fuente completo en el archivo *letritas03.py* de la carpeta *Fuentes*, que acompaña a esta ficha. La misma carpeta contiene el archivo *Texto.txt* que se usó en este ejemplo).

```
# titulo general...
print('Procesamiento de una secuencia de caracteres')
print('Version 3: cargando todo el texto desde un archivo...')
print()

# inicializacion de flags y contadores...
# contador de letras en una palabra...
cl = 0

# contador total de palabras...
ctp = 0

# contador de palabras que empiezan con "p"...
cpp = 0

# contador de palabras que tienen la expresion "ta"...
cta = 0

# flag: la ultima letra vista fue una "t"?...
st = False

# flag: se ha formado la silaba "ta" al menos una vez?...
sta = False

# carga del texto completo, desde el archivo que se supone creado...
print("Procediendo a cargar el archivo Texto.txt...")
m = open("Texto.txt", "rt")
cadena = m.read()
m.close()
input("Archivo cargado... Pulse Enter para procesar el texto...")

# ciclo iterador para procesar el texto...
for car in cadena:
    # contar la letra actual...
    cl += 1

    # fin de palabra?
    if car == ' ' or car == '.':
        # 1.) contar la palabra solo si hubo al menos una letra...
        if cl > 1:
```

```
            ctp += 1

        # 2.) si hubo 'ta' contar la palabra...
        if sta:
            cta += 1

        # reiniciar contador de letras...
        cl = 0

        # reiniciar flags...
        st = sta = False

        # regresar al ciclo ahora mismo...
        continue

    # 2.) deteccion de palabras que empiezan con "p"...
    if cl == 1 and car == 'p':
        cpp += 1

    # 3.) deteccion de expresion "ta"...
    if car == 't':
        st = True
    else:
        if car == 'a' and st:
            sta = True
        st = False

# analisis de los resultados finales...
print('1. Cantidad total de palabras:', ctp)
print('2. Cantidad de palabras que empiezan con "p":', cpp)
print('3. Cantidad de palabras con la expresion "ta":', cta)
```

Bibliografía

[1] Python Software Foundation, "Python Documentation", 2021. [Online]. Available: https://docs.python.org/3/.

[2] V. Frittelli, Algoritmos y Estructuras de Datos, Córdoba: Universitas, 2001.