



UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE COMPUTACIÓN
CÁTEDRA DE COMPUTACIÓN I



Fundamentos de Programación

Con aplicaciones en la Ingeniería

MSc. Ing. Alejandro Bolívar

2023

FUNDAMENTOS DE PROGRAMACIÓN

Con aplicaciones en la Ingeniería

Libro de estudio de la asignatura Computación I.

Fundamentos de Programación con Aplicaciones a la Ingeniería © 2023 by [Alejandro Bolívar](#) is licensed under [CC BY-NC 4.0](#)

Se permite la copia, uso y distribución de este ejemplar, bajo los términos de la licencia **Creative Commons Atribución 4.0**.

Tabla de contenido

1	Tabla de contenido	2
1	Datos Estructurados	5
1.1	Manejo Básico de Cadenas	5
1.2	Listas	12
1.3	Manejo de Tuplas.....	14
1.4	Glosario.....	17
2	Archivos de Texto	19
2.1	Creación y Uso de Archivos de Texto	19
2.2	Campos	19
2.3	Registros	20
2.4	Creación de archivos de Texto.....	20
2.5	Uso de Archivos de Texto en un Programa	21
2.6	Definir el archivo que se usará y de qué modo se usará	21
2.7	Leer ó tomar información en archivos (texto, csv)	22
2.8	Escribir en un archivo	24
2.9	Cerrar el Archivo	25
2.10	Ejemplo de procesamiento de archivos	25
2.11	Lectura hasta el Fin de Archivo	26
2.12	Ejercicios de Archivo de Texto.....	28
2.13	Recorrido de un archivo según su estructura o contenido.....	31
3	Referencias	39

Introducción

Este libro es el producto de la intención de proporcionar a los estudiantes del tercer y cuarto semestre de la Facultad de Ingeniería de la Universidad de Carabobo, en la asignatura Computación I del Departamento de Computación una comprensión sólida y completa de los conceptos fundamentales de la programación enfocado en la resolución de problemas mediante el computador. Partiendo desde la lógica de programación, el estudiante aprenderá a desarrollar algoritmos eficientes y efectivos para resolver problemas complejos en un lenguaje de programación. Además, se abordan temas esenciales como el control de flujo, las estructuras de datos, el diseño modular, y la depuración de código.

Este texto representa una guía detallada y accesible para desarrollar habilidades en una de las disciplinas más relevantes e influyentes de nuestro tiempo. Espero que este libro proporcione la base necesaria para avanzar en su formación como programador, así como también permitirle desarrollar soluciones tecnológicas innovadoras e impactantes.

En el contenido se abarca los temas de datos estructurados y el manejo de archivos de texto, mediante estos temas se ofrece una forma diferente de almacenar y manejar datos.

En <https://github.com/alejandrobolivar/compu-I> puede encontrarse el código de algunos ejemplos.

Unidad 5

Datos Estructurados

Objetivos de la unidad:

En esta unidad aprenderá a utilizar los tipos de datos estructurados, algunos métodos y como recorrer los elementos que estos datos contienen.

Datos Estructurados

1.1 Manejo Básico de Cadenas

Una *cadena* es una sucesión de caracteres encerrada entre comillas (simples o dobles). Python ofrece una serie de operadores y funciones predefinidos que manejan cadenas o devuelven cadenas como resultado [1].

Para seleccionar un carácter según la posición en la cadena, se escribe entre corchetes “[] ” un valor numérico entero que representa el índice del elemento. Por ejemplo:

```
>>> fruta = 'banana'
>>> letra = fruta[1]
>>> print (letra)
```

La expresión `fruta[1]` selecciona el carácter en la posición 1 de la variable `fruta`. La variable `letra` apunta al valor siguiente:

```
a
```

La primera letra de "banana" no es *a*, en computación siempre se empieza a contar desde cero. El contenido almacenado en la posición cero de "banana" es *b*. El valor contenido en la posición 1 es *a*, y el de la posición 2 es la letra *n*.

Sí se requiere la primera letra de una cadena, simplemente escribe 0, o cualquier expresión de valor 0, entre los corchetes:

```
>>> letra = fruta[0]
>>> print (letra)
b
```

Operador + (concatenación de cadenas): acepta dos cadenas como operandos y devuelve la cadena que resulta de unir la segunda a la primera.

```
>>> 'Asignatura = ' + 'Computación I'
'Asignatura= Computación I'
```

Concatenar con datos que no sean cadenas de caracteres: Puedes concatenar con variables que no son cadena simplemente convirtiendo el valor en una cadena usando la función *str()* como se muestra a continuación:

```
str = "This is test number " + str(15)
print (str)
```

Operador * (repetición de cadena): acepta una cadena y un entero y devuelve la concatenación de la cadena consigo misma tantas veces como indica el entero.

```
>>> 'Computación I '*3
'Computación I Computación I Computación I '
```

Operador % (sustitución de marcas de formato): acepta una cadena y una o más expresiones (entre paréntesis y separadas por comas) y devuelve una cadena en la que las marcas de formato (secuencias como %d, %f, etc.) se sustituyen por el resultado de evaluar las expresiones.

```
>>> velocidad = 25.123456
```

Impresión con dos decimales

```
>>> print('velocidad= %.2f' % velocidad)
velocidad= 25.12
```

Impresión sin decimales

```
>>> print('velocidad= %d' % velocidad)
velocidad= 25
```

O para trabajar con cadenas, imprimiendo un nombre

```
>>> nombre = 'Juan'
>>> print('Hola, %s! ' % nombre)
Hola, Juan!
```

En la Tabla 1-1 se muestra los caracteres de formato.

Tabla 1-1. Caracteres de formato de cadena

Character	Description
d or i	Decimal (base 10) integer
f	Floating point number
s	String or any object

c	Single character
u	Unsigned decimal integer
X or x	Hexadecimal integer (upper or lower case)
0	Octal integer
e or E	Floating point number in exponential form
g or G	Like Xf unless exponent < -4 or greater than the precision. If so, acts like Xe or XE
r	repr() version of the object'
X	Use XX to print the percentage character.

Para imprimir dos o más especificadores de argumento, se utiliza una tupla (datos entre paréntesis):

```
>>> nombre = 'Juan'
>>> edad = 23
>>> print('%s tiene %d años. ' % (nombre, edad))
Juan tiene 23 años.
```

O una tupla para el acceso según su posición.

```
>>> tupla = ('Juan',23)
>>> print('%s tiene %d años. ' % (tupla [0], tupla [1]))
Juan tiene 23 años.
```

También funciona para una lista.

```
>>> lista = ['Juan',23]
>>> print('%s tiene %d años. ' % (lista[0], lista[1]))
Juan tiene 23 años.
```

En la Tabla 1-2 se muestra un resumen de los formatos de cadena.

Tabla 1-2. Secuencia de Escape

Sequence	Description
\n	Newline (ASCII LF)
\'	Single quote
\"	Double quote
\\	Backslash
\t	Tab (ASCII TAB)
\b	Backspace (ASCII BS)
\r	Carriage return (ASCII CR)
\xhh	Character with ASCII value hh in hex
\ooo octal	Character with ASCII value ooo in
\f	Form feed (ASCII FF)*
\a	Bell (ASCII BEL)

1.1.1 Longitud de una cadena

La función **len** devuelve el número de caracteres de una cadena:

```
>>> fruta = 'banana'
>>> len(fruta)
6
```

Para obtener la última letra de una cadena realice la siguiente instrucción:

```
longitud = len(fruta)
ultima = fruta[longitud] # ERROR!
```

Eso no funcionaría, provoca un error en tiempo de ejecución *IndexError: string index out of range*. La razón es que no hay una sexta letra en "banana". Como **se empieza a contar por cero**, las seis letras están numeradas del 0 al 5. Para obtener el último carácter se resta 1 a la variable longitud.

```
longitud = len(fruta)
ultima = fruta[longitud-1]
```

De forma alternativa, se puede utilizar índices negativos, que cuentan hacia atrás desde el final de la cadena. La expresión `fruta[-1]` devuelve la última letra. `fruta[-2]` nos da la penúltima, y así sucesivamente.

1.1.2 Porciones de cadenas

Llamamos **porción** a un segmento de una cadena. La selección de una porción es similar a la selección de un carácter:

```
>>> s = 'Pedro, Pablo, y María'
>>> print(s[0:5])
Pedro
>>> print(s[7:12])
Pablo
>>> print(s[15:20])
María
```

El operador `[n:m]` devuelve la parte de la cadena desde el *n*-ésimo carácter hasta el *m*-enésimo", incluyendo el primero, pero excluyendo el último (posición *m*). Este comportamiento contradice a nuestra intuición; tiene más sentido si imagina los índices señalando entre los caracteres, como en el siguiente diagrama:

Fruta → “	b	a	n	a	n	a	“
Índice	0	1	2	3	4	5	

Si omite el primer índice (antes de los dos puntos), la porción comienza al principio de la cadena. Si omite el segundo índice, la porción llega al final de la cadena. Así:

```
>>> fruta = 'banana'
>>> fruta[:3]
'ban'
>>> fruta[3:]
'ana'
```

¿Qué valor devuelve fruta[:]?

Si utilizas un número negativo, el recuento comenzará hacia atrás, como lo ejemplo `fruta[-1]` que imprime el último carácter de la cadena.

1.1.3 Las cadenas son inmutables

Es tentador usar el operador `[]` en el lado izquierdo de una asignación, con la intención de cambiar un carácter en una cadena. Por ejemplo:

```
saludo = 'Hola, mundo'
saludo[0] = 'M' # ERROR!
print(saludo)
```

En lugar de presentar la salida Mola, mundo, este código presenta el siguiente error en tiempo de ejecución *TypeError: object doesn't support item assignment*.

Las cadenas son **inmutables**, lo que significa que no puede cambiar una cadena existente. Lo más que puede hacer es crear una nueva cadena que sea una variación de la original:

```
saludo = 'Hola, mundo'
nuevoSaludo = 'M' + saludo[1:]
print(nuevoSaludo)
```

Aquí la solución es concatenar una nueva primera letra a una porción de saludo. Esta operación no tiene efectos sobre la cadena original.

Se puede manejar cadenas, además, mediante métodos que les son propios (ver Anexo A).

1.1.4 Buscar una subcadena

Puedes buscar una subcadena utilizando el método *find*:

```
str = "welcome to likegeeks website"
print(str.find("likegeeks"))
```

El método *find* imprime la posición de la primera aparición de la cadena *likegeeks* si se encuentra.

Si no encuentra nada, devolverá -1 como resultado.

La función de *find* comienza desde el primer carácter, sin embargo, puede comenzar desde el enésimo carácter de esta manera:

```
str = "welcome to likegeeks website"
print(str.find("likegeeks", 12))
```

Como comenzamos desde el carácter 12, no hay una palabra llamada *likegeeks* desde esa posición, por lo que devolverá -1.

1.1.5 Reemplazar cadenas

Puedes reemplazar una cadena utilizando el método *replace* de la siguiente forma:

```
str = "This website is about programming"
str2 = str.replace("This", "That")
print(str2)
```

Si tiene muchas ocurrencias y deseas reemplazar solo la primera, puede especificar eso:

```
str = "This website is about programming I like this website"
str2 = str.replace("website", "page", 1)
print(str2)
```

Solo la primera palabra fue reemplazada.

1.1.6 Strip de cadenas

Puedes recortar los espacios en blanco de una cadena usando el método *strip* la siguiente forma:

```
str = "  This website is about programming  "
print(str.strip())
```

Puedes quitar solo los de la derecha o izquierda utilizando los métodos *rstrip()* o *lstrip()* respectivamente.

1.1.7 Cambiar entre Caracteres en mayúscula y minúsculas

Puedes cambiar entre mayúsculas y minúsculas los caracteres si necesitas compararlos.

```
str = "welcome to likegeeks"  
print(str.upper())  
print(str.lower())
```

1.1.8 Convertir cadenas en números

Tenemos la función *str()* que convierte el valor a una cadena, pero esta no es la única función de conversión en la programación en Python.

Existen *int()*, *float()*, *long()* y otras funciones de conversión que puede usar.

La función *int()* convierte la entrada a un entero, mientras que la función *float()* convierte la entrada en *float*.

```
str = "10"  
str2 = "20"  
print(str + str2)  
print(int(str) + int(str2))
```

La primera línea de *print* simplemente concatena los dos números sin sumar, mientras que la segunda línea de *print* suma los dos valores e imprime el total.

1.1.9 Contar Cadenas

Puede utilizar las funciones *min()*, *max()* y *len()* para calcular el valor mínimo o máximo de los caracteres o la longitud total de estos.

```
str = "welcome to likegeeks website"  
print(min(str))  
print(max(str))  
print(len(str))
```

1.1.10 Iteraciones sobre cadenas

Puedes iterar sobre las cadenas y manipular cada carácter individualmente de la siguiente forma, este tema se estudiará en estructuras repetitivas:

```
str = "welcome to likegeeks website"
for i in range(len(str)):
    print(str[i])
```

La función **len** () cuenta la longitud de los objetos.

1.1.11 Codificar cadenas

En Python 3, todas las cadenas se almacenan como cadenas Unicode de forma predeterminada, por otro lado, en Python 2 se necesita codificar las cadenas de esta manera:

```
str = "welcome to likegeeks website"
str.encode('utf-8')
```

1.2 Listas

Al igual que una cadena, una lista es una secuencia de valores. En una cadena, los valores son caracteres; en una lista, pueden ser de cualquier tipo. Los valores en las listas reciben el nombre de **elementos**, o a veces **artículos**.

Hay varios modos de crear una lista nueva; el más simple consiste en encerrar los elementos entre corchetes ([y]):

Como es lógico, puedes asignar listas de valores a variables:

```
>>> quesos = ['Cheddar', 'Edam', 'Gouda']
>>> numeros = [17, 123]
>>> lista_anidada = ['spam', 2.0, 5, [10, 20]]
>>> vacia = []
```

1.2.1 Las listas son mutables

La sintaxis para acceder a los elementos de una lista es la misma que para acceder a los caracteres de una cadena, con el operador corchete. La expresión dentro de los corchetes especifica el índice. Recuerda que los índices comienzan por 0:

```
>>> print(quesos[0])  
Cheddar
```

A diferencia de las cadenas, las listas son mutables (pueden mutar), **porque puedes cambiar el orden de los elementos o reasignar un elemento dentro de la lista**. Cuando el operador corchete aparece en el lado izquierdo de una asignación, este identifica el elemento de la lista que será asignado.

```
>>> numeros = [17, 123]  
>>> numeros[1] = 5  
>>> print(numeros)  
[17, 5]
```

El elemento de números cuyo índice es uno, que antes era 123, es ahora 5.

Puedes pensar en una lista como una relación entre índices y elementos. Esta relación recibe el nombre de mapeo o direccionamiento; cada índice “dirige a” uno de los elementos.

Los índices de una lista funcionan del mismo modo que los índices de una cadena:

- ✓ Cualquier expresión entera puede ser utilizada como índice.
- ✓ Si se intenta leer o escribir un elemento que no existe, se obtiene un *IndexError*.
- ✓ Si un índice tiene un valor negativo, cuenta hacia atrás desde el final de la lista.

1.2.2 Recorrer una lista

El modo más habitual de recorrer los elementos de una lista es con un ciclo *for*. La sintaxis es la misma que para las cadenas:

```
for queso in quesos:  
    print(queso)
```

Esto funciona correctamente si sólo se necesita leer los elementos de la lista. Pero si quieres escribir o modificar los elementos, necesitarás los índices. Un modo habitual de hacerlo consiste en combinar las funciones *range* y *len*:

```
for i in range(len(numeros)):
    numeros[i] = numeros[i] * 2
```

Este ciclo recorre la lista y actualiza cada elemento. *len* devuelve el número de elementos de la lista. *range* devuelve una lista de índices desde 0 hasta $n - 1$, donde n es la longitud de la lista. Cada vez que se recorre el ciclo, i obtiene el índice del elemento siguiente. La sentencia de asignación en el cuerpo utiliza i para leer el valor antiguo del elemento y asignarle el valor nuevo.

Un ciclo *for* aplicado a una lista vacía no ejecuta nunca el código contenido en su cuerpo:

```
for x in vacia:
    print('Esto nunca ocurrirá.')
```

1.3 Manejo de Tuplas

1.3.1 Las tuplas son inmutables

Una tupla es una secuencia de valores muy parecida a una lista. Los valores almacenados en una tupla pueden ser de cualquier tipo, y están indexados por valores numéricos enteros que inicia en cero. La diferencia más importante es que las tuplas son inmutables. Las tuplas además son comparables y dispersables (hashables), de modo que las listas de tuplas se pueden ordenar y también es posible usar tuplas como valores para las claves en los diccionarios de Python.

Sintácticamente, una tupla es una lista de valores separados por comas:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

A pesar de que no es necesario, resulta corriente encerrar las tuplas entre paréntesis, lo que ayuda a identificarlas rápidamente dentro del código en Python.

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

Para crear una tupla con un único elemento, es necesario incluir una coma al final:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

Sin la coma, Python trata ('a') como una expresión con una cadena dentro de un paréntesis, que evalúa como de tipo “string”:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

Otro modo de construir una tupla es usar la función interna *tuple*. Sin argumentos, crea una tupla vacía:

```
>>> t = tuple()
>>> print(t)
()
```

Si el argumento es una secuencia (cadena, lista o tupla), el resultado de la llamada a *tuple* es una tupla con los elementos de la secuencia:

```
>>> t = tuple('altramuces')
>>> print(t)
('a', 'l', 't', 'r', 'a', 'm', 'u', 'c', 'e', 's')
```

Dado que *tuple* es una palabra reservada (es el nombre de un constructor), debe evitar utilizarlo como nombre de variable. La mayoría de los operadores de listas funcionan también con tuplas. El operador corchete indexa un elemento:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print(t[0])
'a'
```

Y el operador de sector selecciona un rango de elementos.

```
>>> print(t[1:3])
('b', 'c')
```

Pero si se intenta modificar uno de los elementos de la tupla, se obtiene un error:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

No se pueden modificar los elementos de una tupla, pero se puede reemplazar una tupla con otra:

```
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```


1.3.2 Asignación de tuplas

Una de las características sintácticas del lenguaje Python que resulta única es la capacidad de tener una tupla en el lado izquierdo de una sentencia de asignación. Esto permite asignar varias variables al mismo tiempo cuando se tiene una secuencia en el lado izquierdo.

En este ejemplo se tiene una lista de dos elementos (por lo que se trata de una secuencia), y se asigna los elementos primero y segundo de la secuencia a las variables x e y en una única sentencia.

```
>>> m = ['pásalo', 'bien']
>>> x, y = m
>>> x
'pásalo'
>>> y
'bien'
>>>
```

Python traduce aproximadamente la sintaxis de asignación de la tupla de este modo:

```
>>> m = ['pásalo', 'bien']
>>> x = m[0]
>>> y = m[1]
>>> x
'pásalo'
>>> y
'bien'
>>>
```

Cuando se utiliza una tupla en el lado izquierdo de la sentencia de asignación, se omite los paréntesis. Pero lo que se muestra a continuación es una sintaxis igualmente válida:

```
>>> m = ['pásalo', 'bien']
>>> (x, y) = m
>>> x
'pásalo'
>>> y
'bien'
>>>
```

Una aplicación especialmente ingeniosa de asignación usando una tupla nos permite intercambiar los valores de dos variables en una única sentencia:

```
>>> a, b = b, a
```

Ambos lados de esta sentencia son tuplas, pero el lado izquierdo es una tupla de variables; el lado derecho es una tupla de expresiones. Cada valor en el lado derecho es

asignado a su respectiva variable en el lado izquierdo. Todas las expresiones en el lado derecho son evaluadas antes de realizar ninguna asignación.

La cantidad de variables en el lado izquierdo y la cantidad de valores en el derecho debe ser la misma:

```
>>> a, b = 1, 2, 3
ValueError: too many values to unpack
```

Generalizando más, el lado derecho puede ser cualquier tipo de secuencia (cadena, lista o tupla). Por ejemplo, para dividir una dirección de e-mail en nombre de usuario y dominio, podrías escribir:

```
>>> dir = 'monty@python.org'
>>> nombreus, dominio = dir.split('@')
```

El valor de retorno de *split* es una lista con dos elementos; el primer elemento es asignado a *nombreus*, el segundo a *dominio*.

```
>>> print(nombreus)
monty
>>> print(dominio)
python.org
```

1.4 Glosario

contador: Una variable usada para contar algo, normalmente inicializado a cero e incrementado posteriormente.

incrementar: Aumentar el valor de una variable en una unidad, **decrementar:** Disminuir el valor de una variable en una unidad, **espacio en blanco:** Cualquiera de los caracteres que mueven el cursor sin imprimir caracteres visibles. La constante *string.whitespace* contiene todos los caracteres de espacio en blanco.

índice: Una variable o valor usado para seleccionar un miembro de un conjunto ordenado, como puede ser un carácter de una cadena.

mutable: Un tipo de datos compuesto a cuyos elementos se les puede asignar nuevos valores.

porción: Una parte de una cadena especificada por un intervalo de índices.

recorrer: Realizar de forma iterativa una operación similar sobre cada uno de los elementos de un conjunto.

tipo de datos compuesto: Un tipo de datos en el que los valores están hechos de componentes o elementos que son a su vez valores.

Unidad 6

Archivos de datos secuenciales

En este tema se abarcará las actividades básicas de lectura y escritura de archivos de datos, con estos datos y mediante las herramientas de programación adecuadas se busca resolver los diferentes problemas que se puedan plantear con esta forma sencilla de manejo de archivos.

Objetivos de la Unidad:

En esta unidad aprenderá a manejar los archivos de texto en particular en el lenguaje Python. Esto incluye abrir, cerrar, leer desde y escribir en archivos `.txt`, `.csv`, `.dat`

Archivos de Texto

2.1 Creación y Uso de Archivos de Texto

Un *archivo* o *fichero* es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas *registros* o *artículos*, que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajo denominadas *campos* [2].

El archivo es un medio de almacenamiento separado del programa, que guarda en el mismo una gran cantidad de información. El uso de un archivo para guardar datos de forma externa es recomendable en procesos que no necesiten una organización muy compleja de los datos a gestionar ya que en tal caso lo mejor sería utilizar un programa gestor de base de datos, que ya incorpora, de modo mucho más optimizado, los algoritmos y procesos específicos para el tratamiento de la información.

2.2 Campos

Un campo (ver Figura 2-1) es un ítem o elemento de datos elementales, tales como un nombre, número de empleados, ciudad, número de identificación, etc. Un campo está caracterizado por su tamaño o longitud y su tipo de datos (cadena de caracteres, entero, lógico, etcétera.). Los campos pueden incluso variar en longitud. En la mayoría de los lenguajes de programación los campos de longitud variable no están soportados y se suponen de longitud fija [2].

Figura 2-1. Campos de un registro.

Nombre	Dirección	Fecha de nacimiento	Estudios	Salario	Trienios
--------	-----------	---------------------	----------	---------	----------

Registros

Un campo es la unidad mínima de información de un registro.

Los datos contenidos en un campo se dividen con frecuencia en *subcampos*; por ejemplo, el campo fecha se divide en los subcampos día, mes, año.

<i>Campo</i>	0	7	0	7	1	9	9	5
<i>Subcampo</i>	Día		Mes		Año			

Los rangos numéricos de variación de los subcampos anteriores son:

$$1 \leq \text{día} \leq 31$$

$$1 \leq \text{mes} \leq 12$$

$$\leq \text{año} \leq 1987$$

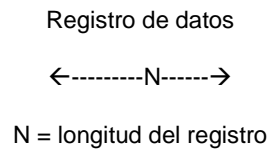
2.3 Registros

Un registro es una colección de información, normalmente relativa a una entidad particular [2]. Un registro es una colección de campos lógicamente relacionados (ver Figura 2-2), que pueden ser tratados como una unidad por algún programa. Un ejemplo de un registro puede ser la información de un determinado empleado que contiene los campos de nombre, dirección, fecha de nacimiento, estudios, salario, trienios, etc.

Los registros pueden ser todos de longitud fija; por ejemplo, los registros de empleados pueden contener el mismo número de campos, cada uno de la misma longitud para nombre, dirección, fecha, etc. También pueden ser de longitud variables.

Los registros organizados en campos se denominan registros lógicos.

Figura 2-2. Registro.



2.4 Creación de archivos de Texto

Para crear un archivo de texto se puede utilizar el editor de texto *Block de Notas* o *Note Pad*, en el mismo se escribirá la lista de valores o datos que se desean procesar

separándolos entre sí por una coma, el orden de los datos debe corresponder con la estructura definida para los mismos, sin líneas vacías, ni títulos.

Suponga que se dice que se tienen los datos de un conjunto de personas donde para cada persona se tiene *Nombre*, *Cédula* y *Nota*, el archivo correcto sería el mostrado a la derecha, el cual se guardará como *datos.dat*.

2.5 Uso de Archivos de Texto en un Programa

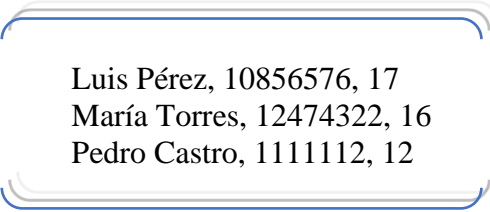
Para utilizar un archivo de texto en un programa se debe:

- a. Definir el archivo que se usará y de qué modo se usará.
- b. Usar el archivo según lo definido (Leer (*read*, *readlines*, *readline*) o Guardar (*write*)).
- c. Cerrar el archivo.

2.6 Definir el archivo que se usará y de qué modo se usará

Los archivos en Python son objetos de tipo *file* creados mediante la función *open* (abrir). Esta función toma como parámetros una cadena con la ruta del archivo a abrir, que puede ser relativa o absoluta; una cadena opcional indicando el modo de acceso (si no se especifica se accede en modo lectura) y, por último, un entero opcional para especificar un tamaño de *buffer* distinto del utilizado por **defecto**.

El modo de acceso que es un parámetro opcional, puede ser cualquier combinación lógica de los siguientes modos:



Luis Pérez, 10856576, 17
María Torres, 12474322, 16
Pedro Castro, 1111112, 12

1. **'r'**: read, lectura. Abre el archivo en **modo lectura**, es el modo por defecto. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo `IOError`.
2. **'w'**: write, escritura. Abre el archivo en modo escritura. Si el archivo **no existe se crea. Si existe**, sobrescribe el contenido.
3. **'a'**: append, añadir. Abre el archivo en modo escritura. Se diferencia del modo **'w'** **en que en este** caso no se sobrescribe el contenido del archivo, sino que se comienza a **escribir al final del archivo**.
4. **'b'**: **binary, binario**.
5. **'+'**: permite lectura y **escritura simultáneas**.
6. **'u'**: universal newline, saltos de línea **universales**. **Permite trabajar** con archivos que tengan un formato para los saltos de línea que no coincide con el de la plataforma actual (en Windows se utiliza el carácter CR LF, en Unix LF y en Mac OS CR).

```
arch = open(nombre_del_archivo, modo_de_acceso)
```

Por ejemplo, para abrir el archivo *agenda.txt* en modo de lectura, se escribe la siguiente instrucción.

```
arch = open("agenda.txt", 'r')
```

Tras crear la variable (objeto) que representa el **archivo mediante la función *open*** se puede realizar las operaciones de lectura/escritura pertinentes utilizando los métodos **del objeto que veremos en las siguientes secciones**.

Una vez se termine de trabajar con el archivo, se debe cerrarlo utilizando el método *close*.

```
arch.close()
```

2.7 Leer ó tomar información en archivos (texto, csv)

La función *open* crea una referencia a un archivo (usualmente llamado *file handle*) que se usa para leer los datos:

```
open(filename[, mode[, bufsize]])
```

Ejemplo de generación de *file handler* asociado al archivo *mi_archivo.txt*.

```
arch = open('mi_archivo.txt', 'r')
```

La 'r' indica "modo de lectura" y es el modo por defecto (por lo que se podría obviar la 'r').

Sobre el *file handle* (manejador de archivo ***arch***) se puede:

- a. read(n): Lee n bytes, por defecto lee el archivo entero.
- b. readline(): Devuelve *str* con una sola línea.
- c. readlines(): Devuelve una lista con una cadena como elemento por cada línea del archivo.

El Código 2-1 muestra diferentes códigos para realizar la lectura de un archivo de texto.

Código 2-1. Distintas maneras de leer el contenido de un archivo.

```
# script 1
arch = open('archivo.txt')
contenido = arch.read()
print (contenido)

# script 2
arch = open('archivo.txt')
contenido = arch.readlines()
print (contenido)

# script 3
contenido = ''
arch = open('archivo.txt')
while True:
    line = arch.readline()
    contenido += line
    if line == '':
        break
print(contenido)

# Para todos los casos:
arch.close()
```

El resultado de la lectura es el siguiente:

```
Luis Pérez, 10856576, 17
María Torres, 12474322, 16
Pedro Castro, 1111112, 12
['Luis Pérez, 10856576, 17\n', 'María Torres, 12474322, 16\n', 'Pedro Castro, 1111112, 12']
Luis Pérez, 10856576, 17
María Torres, 12474322, 16
```


Pedro Castro, 1111112, 12

Acceso secuencial al contenido de un archivo:

```
arch = open('archivo.txt')
contenido = ''
for linea in arch:
    contenido += linea
fh.close()
```

Leyendo con 'with' (Python 2.6 en adelante)

Manera genérica:

```
with <Expresión> as <Variable>:
    <Código>
```

Ejemplo:

```
with open('archivo.txt') as arch:
    for line in arch:
        print(line)
```

La ventaja de *with* es que, al salir del bloque, se ejecuta un método especial que cierra el archivo automáticamente.

En el caso de leer un archivo con un *encoding* en particular:

```
import codecs
f = codecs.open('arch.txt', encoding='utf-8')
```

2.8 Escribir en un archivo

De la misma forma que para la lectura, existen dos formas distintas de escribir a un archivo.

Mediante cadenas:

```
archivo.write(cadena)
```

O mediante listas de cadenas:

```
archivo.writelines(lista_de_cadenas)
```

Así como la función *read* devuelve las líneas con los caracteres de fin de línea “\n”, será necesario agregar los caracteres de fin de línea a las cadenas que se vayan a escribir en el archivo.

Código 2-2. genera_saludo.py: Genera el archivo saludo.py

```
saludo = open("saludo.py", "w")
saludo.write("""
print "Hola Mundo"
""")
saludo.close()
```

El ejemplo que se muestra en el Código 2-1 contiene un programa Python que a su vez genera el código de otro programa Python.

Modos de escritura:

w: Write, graba un archivo nuevo, si existe, lo borra.

a: Append (agregar), agrega información al final de un archivo preexistente. Si no existe, crea uno nuevo.

Ejemplo:

```
arch = open('/home/yo/archivo.txt', 'w')
arch.write('1\n2\n3\n4\n5\n')
arch.close()
```

2.9 Cerrar el Archivo

Al terminar de trabajar con un archivo, es recomendable cerrarlo, por diversos motivos: en algunos sistemas los archivos sólo pueden ser abiertos un programa a la vez; en otros, lo que se haya escrito no se guardará realmente hasta no cerrar el archivo; o el límite de cantidad de archivos que puede manejar un programa puede ser bajo, etc.

Para cerrar un archivo simplemente se debe ejecutar a:

```
arch.close()
```

ADVERTENCIA: Es importante tener en cuenta que cuando se utilizan funciones como *archivo.readlines()*, se está cargando en memoria el archivo completo. Siempre que una instrucción cargue un archivo completo en memoria debe tenerse cuidado de utilizarla sólo con archivos pequeños, ya que de otro modo podría agotarse la memoria de la computadora.

2.10 Ejemplo de procesamiento de archivos

Por ejemplo, para mostrar todas las líneas de un archivo, precedidas por el número de línea, podemos hacerlo como en el Código 2-3.

Código 2-3. numera_lineas.py: Imprime las líneas de un archivo con su número

```
arch = open("archivo.txt")
i = 1
for linea in arch:
    linea = linea.rstrip("\n")
    print (" %4d: %s" % (i, linea))
    i+=1
arch.close()
```

La llamada a *rstrip* es necesaria ya que cada línea que se lee del archivo contiene un fin de línea y con la llamada a *rstrip("\n")* se remueve.



Precauciones al trabajar con archivos

Debes cerrar todos los archivos tan pronto termines de trabajar con ellos. Si la aplicación finaliza normalmente, el sistema operativo cierra todos los archivos abiertos, así que no hay pérdida de información. Esto es bueno y malo a la vez. Bueno porque si olvidas cerrar un archivo y tu programa está, por lo demás, correctamente escrito, al salir todo quedará correctamente almacenado; y malo porque es fácil que te relajes al programar y olvides la importancia que tiene el correcto cierre de los archivos. Esta falta de disciplina hará que acabes por no cerrar los archivos cuando hayas finalizado de trabajar con ellos, pues “ellos solos ya se cierran al final”.

El riesgo de pérdida de información inherente al trabajo con archivos hace que debas ser especialmente cuidadoso al trabajar con ellos. Es deseable que los archivos permanezcan abiertos el menor intervalo de tiempo posible. Si una función o procedimiento actúa sobre un archivo, esa subrutina debería abrir el archivo, efectuar las operaciones de lectura/escritura pertinentes y cerrar el archivo. Sólo cuando la eficiencia del programa se vea seriamente comprometida, deberás considerar otras posibilidades.

Es más, deberías tener una política de copias de seguridad para los archivos de modo que, si alguna vez se corrompe uno, puedas volver a una versión anterior tan reciente como sea posible.

2.11 Lectura hasta el Fin de Archivo

A modo de ejemplo, el siguiente programa Python abre un archivo y lo copia en otro

```
f = open('origen.txt')
g = open('destino.txt', 'w')
for linea in f:
    g.write(linea)
```

```
g.close()
f.close()
```

Lo interesante aquí es el ciclo *"for linea in f"*. Esta es una forma de recorrer un archivo de texto, obteniendo una línea cada vez.

El archivo se puede leer con *f.readLine()* que nos da una línea cada vez, incluyendo el retorno de carro `\n` al final. Cuando se llega al final de archivo devolverá una línea vacía. Una línea en blanco en medio del archivo nos sería devuelta como un `"\n"`, no como una línea vacía `""`. El siguiente ejemplo hace la copia del archivo leyendo con *readline()* y en un ciclo hasta fin de archivo.

```
f = open('origen.txt')
g = open('destino.txt', 'w')
linea = f.readline()
while linea != '':
    g.write(linea)
    linea = f.readline()
g.close()
f.close()
```

Por ejemplo, para mostrar todas las líneas de un archivo, precedidas por el número de línea, podemos hacerlo como en el Código 2-4.

Código 2-4 numera_lineas.py: Imprime las líneas de un archivo con su número

```
archivo = open("archivo.txt")
i = 1
for linea in archivo:
    linea = linea.rstrip("\n")
    print(" %4d: %s" % (i, linea))
    i+=1
archivo.close()
```

El script anterior muestra:

```
1: Luis Pérez, 10856576, 17
2: María Torres, 12474322, 16
3: Pedro Castro, 1111112, 12
```

La llamada a *rstrip* es necesaria ya que cada línea que se lee del archivo contiene un fin de línea y con la llamada a *rstrip("\n")* se remueve.

Otra opción para hacer exactamente lo mismo sería utilizar la función de Python *enumerate*(secuencia). Esta función devuelve un contador por cada uno de los elementos que

se recorren, puede usarse con cualquier tipo de secuencia, incluyendo archivos. La versión equivalente se muestra en el Código 2-5.

Código 2-5 numera_lineas2.py: Imprime las líneas de un archivo con su número

```
arch = open("archivo.txt")
for i, linea in enumerate(arch):
    linea = linea.rstrip("\n")
    print (" %4d: %s" % (i, linea))
arch.close()
```

2.12 Ejercicios de Archivo de Texto

Ejercicio 1. Sondeo de opinión

Durante un sondeo de opinión, se aplicó una encuesta anónima a un conjunto de N personas, donde por cada persona encuestada, se registró en un archivo de nombre *encuesta.txt* lo siguiente para cada encuestado: **Nombre, Género (M=Masculino, F=Femenino), Edad y el número del canal de televisión que más le gusta**. Se conoce cuantas personas fueron encuestadas, esta información se registró en la primera línea del archivo *encuesta.txt*. Desarrolle un programa (ver Código 2-6) que indique el género y el canal favorito de televisión de la persona con la mayor edad registrada.

Ejemplo del archivo *encuesta.txt*

```
5
Sofía González, F, 24, 65
Luisa Jiménez, F, 48, 36
Carlos Montes, M, 72, 54
Julia Torres, F, 51, 36
Juan García, M, 36, 7
```

Código 2-6. Sondeo.py

```
arch=open("encuesta.txt", "r")
# Lectura de datos
n = int(arch.readline()) # Lee de la primera línea el número de encuestados
print(n)
asignado = False # Sin el primer mayor
for registro in arch:
    linea = registro.split(',') # Lista con los campos del registro
    print (linea)
```

Archivos de Texto
Ejercicios de Archivo de Texto

```
nombre = linea[0]      # nombre de la persona
print(nombre)
genero = linea[1]      # genero M=Masculino, F=Femenino
print(genero)
edad = int(linea[2])   # Edad de la Persona
print(edad)
canal = int(linea[3])  # Canal favorito
print(canal)
# Proceso para determinar el mayor y guardar los datos requeridos
if not(asignado):
    mayor = edad
    mgenero = genero
    mCanal = canal
    asignado = True     # Se tiene el primer mayor
else: # a partir del segundo se compara
    if edad > mayor:    # si el dato a procesar excede al mayor
        mayor = edad
        mgenero = genero
        mCanal = canal

arch.close() # cerrar archivos

# Resultados
print("Género de la persona con Mayor Edad:" , mgenero)
print("Canal favorito de la persona con Mayor Edad:" , mCanal)
#Fin del programa
input("Pulse una tecla para finalizar...") # Pausa
```

Ejecución del programa

```
5
['Sofía González', ' F', ' 24', ' 65\n']
Sofía González
F
24
65
['Luisa Jiménez', ' F', ' 48', ' 36\n']
Luisa Jiménez
F
48
36
['Carlos Montes', ' M', ' 72', ' 54\n']
Carlos Montes
M
72
54
['Julia Torres', ' F', ' 51', ' 36\n']
Julia Torres
F
51
36
```

```
[ 'Juan García', ' M', ' 36', ' 7' ]
Juan García
M
36
7
Género de la persona con Mayor Edad: M
Canal favorito de la persona con Mayor Edad: 54
Pulse una tecla para finalizar...
```

Ejercicio 2. Definitiva de alumnos.

Dado el archivo *datos.dat* el cual contiene la información de los estudiantes de una sección *Nombre, Cédula y Nota definitiva*, desarrolle un programa (ver Código 2-7) que genere dos archivos uno con la información de los aprobados y otra con los reprobados de nombres *aprobados.dat* y *reprobados.dat* respectivamente.

Nota: No se conoce el número de estudiantes almacenados.

Código 2-7 DefinitivaAlumnos.py

```
# Apertura de los archivos

# Archivo de lectura
arch1 = open('datos.dat', 'r')
# Archivos a generar
arch2 = open('aprobados.dat', 'w')
arch3 = open('reprobados.dat', 'w')

# Recorrido del archivo
for registro in arch1:
    # Devuelve una lista con los campos del registro
    linea = registro.split(',')
    print(linea)

    #Procesar elementos:
    nombre = linea[0]      # Nombre del estudiante
    print(nombre)
    cedula = linea[1]      # Cédula del estudiante
    print(cedula)
    nota = int(linea[2])   # Nota del estudiante
    print(nota)

    if nota >= 10: # Aprobado
        arch2.write(nombre + ', ' + cedula + '\n')
    else: # Reprobado
        arch3.write(nombre + ', ' + cedula + '\n')

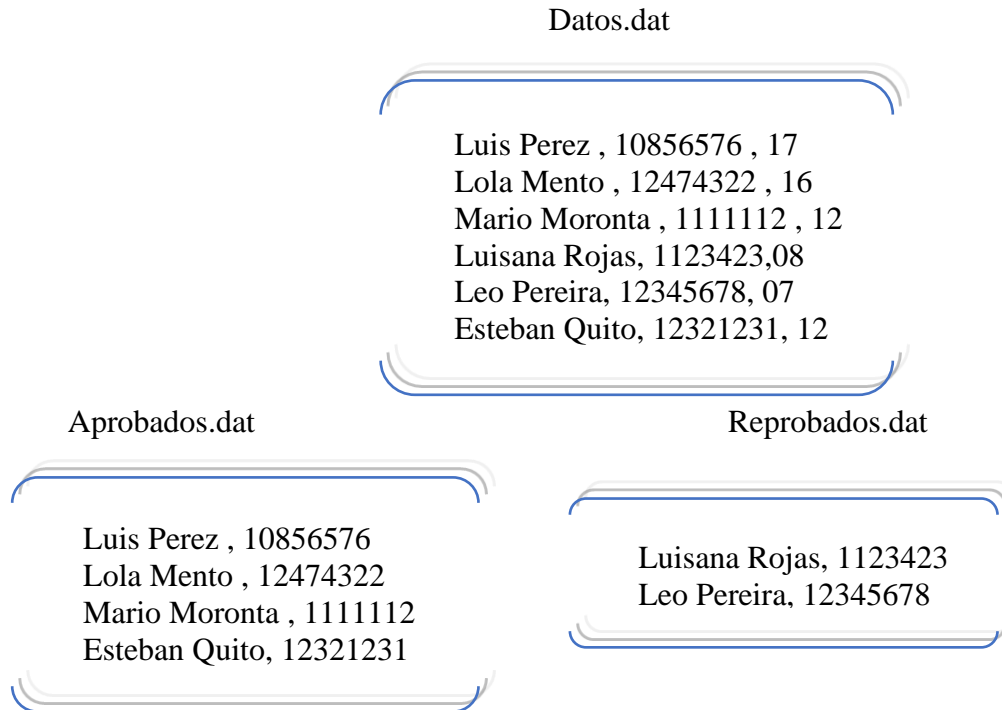
# Cerrar archivos
arch1.close()
```

Archivos de Texto

Recorrido de un archivo según su estructura o contenido.

```
arch2.close()
arch3.close()

# Escritura de resultados
print('Los resultados están en los archivos aprobados y reprobados.dat en el
directorio de trabajo')
```



2.13 Recorrido de un archivo según su estructura o contenido.

2.13.1 Lista sencilla

El recorrido del archivo será en forma secuencial, los archivos presentan una estructura similar en todas sus líneas o registros y se desea recorrer todo el archivo, se puede utilizar las instrucciones del Código 2-8; **Error! No se encuentra el origen de la referencia..**

Considere el siguiente archivo para realizar su lectura e impresión por pantalla.

datos_transito.txt

Maria,1,15,25
Pedro,2,9,50
Juan,2,14,20
Pablo,2,14,30
Moises,2,1,80

Código 2-8. Lectura e impresión de las líneas del archivo *datos_transito.txt*:

```
# Manejo de archivos
with open("datos_transito.txt", 'r') as arch1:
    # Proceso
    for registro in arch1:
        # lectura de datos
        lista = registro.split(',')

        nombre = lista[0]
        genero = int(lista [1])
        hora = int(lista [2])
        velocidad = float(lista [3])

        print('%s, %d, %d, %.2f' % (nombre, genero, hora, velocidad))
```

observe que la estructura **for registro in arch1:**, realiza el recorrido sobre cada línea del archivo de manera automática, la variable **registro** devuelve el contenido de la línea en la que se encuentra el apuntador de registro como tipo **str**, la función Split convierte en lista el contenido de la variable **registro**.

2.13.2 Lectura parcial o total del archivo

Ocasionalmente se puede presentar el caso de la lectura parcial del archivo (ver Código 2-9), en este caso se indica en la primera línea del archivo la cantidad de líneas a procesar. En el archivo *redes.txt* se tiene la información de *NOMBRE*, *GÉNERO*, *EDAD Y RED SOCIAL MÁS UTILIZADA*[1-4] y la cantidad de registros a procesar en la primera línea del archivo.

redes.txt

```
9
sergio,M,25,1
rosa,F,30,2
isabella,F,20,4
michelle,F,18,1
alfonzo,M,23,3
pedro,M,19,4
josefina,F,35,3
aurelio,M,40,2
yolanda,F,15,1
```

La instrucción: **contenido = arch1.readlines()**, realiza la lectura de todo el archivo *redes.txt*, la variable **contenido** es una lista cuyos elementos son cadenas que contienen la información de cada línea del archivo, al final de cada elemento incluye un salto de línea **\n**,

en caso de requerir eliminar el carácter de salto de línea, se emplea la función **strip('\n')**. Esto implica que siempre se conoce tanto la cantidad de registros como la cantidad de datos por registro del archivo.

La variable **w** en la instrucción: **w = int(contenido[linea])**, recibe el valor correspondiente a la cantidad de líneas a leer del archivo, dicho valor se debe validar que sea menor a la cantidad total de líneas del archivo; como se encuentra en la primera línea del archivo, el valor de la variable **linea** es cero.

Código 2-9. Lectura de las n primeras líneas de un archivo

```
with open('redes.txt','r') as arch1:
    contenido = arch1.readlines()
    linea = 0
    w = int(contenido[linea])
    linea += 1

    for _ in range(1, w + 1):
        lista = contenido[linea].split(',')
        linea += 1

        nombre = lista[0]
        genero = lista[1]
        edad = int(lista[2])
        red = int(lista[3])

        print('%s, %s, %d, %d' % (nombre, genero, edad, red))
```

La estructura de control **for _ in range(1, w + 1):**, ejecuta **w** iteraciones para leer las líneas del archivo, se utiliza el guion de subrayado “_” en lugar del índice la estructura de control, debido a que dicho índice no será utilizado en el bloque de la estructura.

La variable **linea** se incrementará su valor cada vez que se pase por una línea del archivo de datos, la instrucción: **lista = contenido[linea].split(',')** devuelve una lista con elementos de tipo *str* del contenido ubicado en la posición **linea** del archivo.

2.13.3 Lista sencilla con una cantidad conocida de datos a leer por registro.

Otra estructura de archivo ofrece en la primera línea del archivo la cantidad de grupos de elementos a leer por cada línea del archivo (ver Código 2-10), por ejemplo, en el archivo **partidos.txt** por cada país se tiene un registro de los goles a favor y goles en contra, esto significa que cada par de datos ofrece la información correspondiente a cada partido:

partidos.txt

Recorrido de un archivo según su estructura o contenido.

```
3
Italia, 1, 1, 1, 1, 2, 0
España, 1, 1, 5, 0, 2, 0
Portugal, 0, 1, 3, 2, 2, 1
Francia, 1, 1, 2, 0, 0, 2
Alemania, 2, 0, 2, 1, 2, 1
```

La estructura general es la siguiente:

Nombre del País, GFpartido1, GCpartido1, GFpartido2, GCpartido2, ... GFpartidoN, GCpartidoN

Para este caso en particular se está tomando como cantidad de partidos 3 (dato de la primera línea), es decir se tienen 6 datos correspondientes a los goles a favor y los goles en contra para cada partido disputado.

El Código 2-10 realiza la lectura del archivo *partidos.txt* en el cual se tiene en la primera línea la cantidad de partidos realizados por país, esto significa que por cada partido se debe leer dos datos correspondientes a la cantidad de goles a favor y goles en contra respectivamente.

Código 2-10. Leer un archivo con n grupos de datos en cada línea del archivo.

```
# Manejo de Archivos
with open("partidos.txt", "r") as arch1:
    contenido = arch1.readlines()
    linea = 0
    cp = int(contenido[linea]) # cantidad de partidos
    linea += 1 # Posición del siguiente elemento

    # Ciclo de lectura
    for _ in range(1, len(contenido)): # Ciclo que procesa los países

        lista = contenido[linea].split(',')
        linea += 1

        nom = lista[0]
        print(nom)

        for i in range(1, cp + 1):
            gf = int(lista[2 * i - 1])
            gc = int(lista[2 * i])
            print('%d, %d' % (gf, gc))
```

Archivos de Texto
Recorrido de un archivo según su estructura o contenido.

El primer ciclo ***for _ in range(1, len(contenido))***: se encarga de recorrer todo el archivo de datos y el ciclo interno ***for i in range(1, cp + 1)***: recorre en cada línea la información correspondiente a cada partido, compuesto por dos datos que son los goles a favor y goles en contra del partido.

La variable lista de tipo ***list***, contiene en el índice cero el nombre del equipo, en las posiciones impares $2 * i - 1$ los goles a favor y en las posiciones pares $2 * i$ los goles en contra.

2.13.4 La lista externa e interna de tamaño conocido.

Los archivos también pueden estar estructurados de manera anidada, es decir, presentar dos estructuras diferentes internamente, en este caso se presenta el archivo ***ordenes.txt*** con una cantidad de elementos conocidos a procesar en su primera línea, y de cada orden de compra se tiene la siguiente información: Número de la orden, Nombre del cliente y la cantidad de órdenes del cliente, seguidamente se escribe la lista interna correspondiente a varios renglones de especificaciones: Área a cubrir [m²], dimensiones de la baldosa deseada (largo y ancho [cm]) y precio por caja de la baldosa deseada.

ordenes.txt

```
2
21102010, Juan Aguilar, 3
25500.22, 10, 20, 12
260.35, 10, 10, 10.3
500.52, 30, 30, 15.9
22102010, Maria Silva, 4
120.36, 10, 10, 10.3
150.26, 20, 25, 20.3
613.21, 10, 20, 36.2
145.20, 12, 15, 25.6
```

El siguiente código (ver Código 2-11) realiza el manejo de la información del archivo mediante dos ciclos ***for*** que recorren la información de la lista externa y la información de la lista interna con tamaños conocidos (n y m respectivamente).

Código 2-11. Lectura de n (conocido) elementos de una lista externa y m elementos de una lista interna.

```
with open('ordenes.txt','r') as arch1:
    contenido = arch1.readlines()
    n = int(contenido[0]) # cantidad de elementos de la lista externa
    linea = 1 # Posición del primer elemento de la lista externa

    for _ in range(1, n + 1): # CICLO QUE PROCESA LAS ORDENES (lista
externa)
        # LECTURA DE LAS ORDENES
        listaext = contenido[linea].split(',')
        linea += 1 # se incrementa el contador de líneas de contenido
        noc = int(listaext[0])
        nombre = listaext[1]
        m = int(listaext[2]) # cantidad de elementos de la lista interna

        print('%d, %s, %d' % (noc, nombre, m))

        for _ in range(m): # CICLO QUE PROCESA LOS RENGLONES (lista interna)
            listaint = contenido[linea].split(',')
            linea += 1 # se incrementa el contador de líneas de contenido
            area = float(listaint[0])
            l = float(listaint[1])
            a = float(listaint[2])
            monto = float(listaint[3])
            print('%.2f, %.2f, %.2f, %.2f' % (area, l, a, monto))
```

2.13.5 Lista externa de tamaño desconocido y lista interna de tamaño conocido.

En caso de no estar incluido el tamaño (*n*) de la lista externa como se aprecia en el siguiente archivo.

ordenes.txt

```
21102010, Juan Aguilar, 3
25500.22, 10, 20, 12
260.35, 10, 10, 10.3
500.52, 30, 30, 15.9
22102010, Maria Silva, 4
120.36, 10, 10, 10.3
150.26, 20, 25, 20.3
613.21, 10, 20, 36.2
145.20, 12, 15, 25.6
```

En este caso se puede leer el archivo (ver Código 2-12) utilizando un ciclo *while* condicionado mediante una variable contador denominada *linea* encargada de contar hasta el final del archivo y un ciclo *for* para el recorrido de la lista interna (*m* es conocido), observe que el contador línea se incrementa en ambos ciclos:

Código 2-12. Lectura de n (desconocido) elementos de una lista externa y m elementos de una lista interna

```
with open('ordenes 2.txt','r') as arch1:
    contenido = arch1.readlines()
    # cantidad de elementos de la lista externa desconocida
    linea = 0 # Posición del primer elemento de la lista externa

    while linea < len(contenido): # Ciclo que procesa las ordenes (lista
externa)
        # Lectura de las ordenes
        listaext = contenido[linea].split(',')
        linea += 1 # se incrementa el contador de líneas de contenido
        noc = int(listaext[0])
        nombre = listaext[1]
        m = int(listaext[2]) # cantidad de elementos de la lista interna
        print('%d, %s, %d' % (noc, nombre, m))

        for _ in range(m): # CICLO QUE PROCESA LOS RENGLONES (lista interna)
            listaint = contenido[linea].split(',')
            linea += 1 # se incrementa el contador de líneas de contenido
            area = float(listaint[0])
            l = float(listaint[1])
            a = float(listaint[2])
            monto = float(listaint[3])
            print('%.2f, %.2f, %.2f, %.2f' % (area, l, a, monto))
```

2.13.6 Lista externa e interna de tamaño desconocido.

Sí se desconoce el tamaño de la lista externa e interna, pero se dispone de una variable *centinela* en cada línea de la lista interna como se muestra en el siguiente archivo de datos *recepción.txt*. En el mismo se presenta una estructura con dos listas bien demarcadas: en una línea la **Fecha de Recepción del paquete** y en las siguientes líneas **nombre del cliente, ciudad destino, peso del paquete, dimensiones del paquete: Largo, Ancho y Alto del paquete (metros) y la centinela**; hasta llegar a la siguiente fecha de recepción del paquete.

recepción.txt

```
15/02/2018
Lola mento, Pto La Cruz, 3, 0.25, 0.25, 0.5, 0
Alan Brito, San Cristobal, 4.5, 1.5, 1.0, 2.0, 0
Pepe Trueno, Caracas, 25, 1.5, 0.5, 0.25, 1
18/02/2019
Sere Brito, Maracay, 3.5, 1.5, 1.0, 2.0, 0
Lola mento, Caracas, 3, 0.5, 0.5, 1.0, 0
Alan Brito, San Cristobal, 4.5, 1.5, 1.0, 2.0, 0
```

Recorrido de un archivo según su estructura o contenido.

Pepe Trueno, Pto La Cruz, 25, 1.5, 0.5, 0.25, 1
19/02/2019
Sere Brito, Maracay, 3.5, 1.5, 1.0, 2.0, 0
Lola mento, Pto La Cruz, 3, 0.5, 0.5, 1.0, 0
Alan Brito, Barquisimeto, 4.5, 0.5, 0.25, 0.5, 0
Pepe Trueno, Caracas, 25, 1.5, 0.5, 0.25, 1

El siguiente código (ver Código 2-13) permite procesar este caso.

Código 2-13 Código para leer todas las líneas de la lista externa y de la lista interna del archivo *recepción.txt*.

```
with open('recepcion.txt', 'r') as arch1:
    # registros, contiene toda la información del archivo
    contenido = arch1.readlines()
    linea = 0

    while linea < len(contenido):
        # fecha_recepcion contiene ['15', '02', '2018\n']
        fecha_recepcion = contenido[linea].split('/')
        linea += 1
        cent = 0
        print(fecha_recepcion[0], '/', fecha_recepcion[1], '/',
              fecha_recepcion[2].strip('\n'))

        while cent == 0:
            # registro contiene
            # ['Lola mento', ' Pto La Cruz', ' 3', ' 0.25', ' 0.25', ' 0.5', ' 0\n']
            lista_int = contenido[linea].split(',')
            cliente = lista_int[0]
            destino = lista_int[1].strip()
            paquete = [float(lista_int[2]), float(lista_int[3]),
                       float(lista_int[4]), float(lista_int[5])]
            cent = int(lista_int[6])
            linea += 1
            print(cliente, destino, paquete[0], paquete[1], paquete[2],
                  paquete[3], cent)
```

- [1] A. Marzal Varó, I. Gracia Luengo y P. García Sevilla, Introducción a la Programación con python 3, Castellón de la Plana: Publicacions de la Universitat Jaume I, 2014.
- [2] L. Joyanes Aguilar, Fundamentos de programación: Algoritmos, estructura de datos y objetos, 4a. ed., Madrid: McGraw-Hill, 2008.
- [3] A. Soto Suarez y M. Arriagada Benítez, Introducción a la Programación con Python, Pontificia Universidad Catolica de Chile, 2015.
- [4] M. J. Mathieu, Introducción a la programación, PRIMERA EDICIÓN EBOOK ed., México: Grupo Editorial Patria, S.A. de C.V, 2014.
- [5] P. Gomis, Fundamentos de Programación en Python, Catalunya: Universitat Politècnica de Catalunya, 2018.
- [6] A. Dasso y A. Funes, Introducción a la Programación, Universidad Nacional de San Luis, 2014.
- [7] E. Bahit, Python para Principiantes, Buenos Aires, 2012.

- [8] E. Bahit, Introduccion al Lenguaje Python, Buenos Aires, 2018.
- [9] J. M. R. Torres, Manual básico, iniciación a Python 3, 2020.