Introducción al lenguaje Python híbrido para la ciencia de datos (Python Data Science)

Book · September 2018

1 author:

Eugenia Bahit
Bahit & Bahit Ltd
25 PUBLICATIONS **1** CITATION
SEE PROFILE

EUGENIA BAHIT

CIENCIA DE DATOS CON PYTHON

MATERIAL DE ESTUDIO

Informes e inscripción:
Curso: http://escuela.eugeniabahit.com | **Certificaciones**: http://python.laeci.org

SUMARIO

MÉTODOS DE MANIPULACIÓN DE VARIABLES

En Python, toda variable se considera un *objeto*. Sobre cada objeto, pueden realizarse diferentes tipos de acciones denominadas *métodos*. Los métodos son funciones pero que se desprenden de una variable. Por ello, se accede a estas funciones mediante la sintaxis:

```
variable.funcion()
```

En algunos casos, estos métodos (funciones de un objeto), aceptarán parámetros como cualquier otra función.

```
variable.funcion(parametro)
```

MANIPULACIÓN DE CADENAS DE TEXTO

A continuación, se verán los principales métodos que pueden aplicarse sobre una cadena de texto, organizados por categorías.

MÉTODOS DE FORMATO

CONVERTIR A MAYÚSCULA LA PRIMERA LETRA

Método: capitalize()
Retorna: una copia de la cadena con la primera letra en mayúsculas

```
>>> cadena = "bienvenido a mi aplicación"
>>> resultado = cadena.capitalize()
>>> resultado
Bienvenido a mi aplicación
```

CONVERTIR UNA CADENA A MINÚSCULAS

Método: lower()
Retorna: una copia de la cadena en minúsculas

```
>>> cadena = "Hola Mundo"
>>> cadena.lower()
```

```
hola mundo
```

CONVERTIR UNA CADENA A MAYÚSCULAS

Método: upper()
Retorna: una copia de la cadena en mayúsculas

```
>>> cadena = "Hola Mundo"
>>> cadena.upper()
HOLA MUNDO
```

CONVERTIR MAYÚSCULAS A MINÚSCULAS Y VICEVERSA

Método: swapcase()
Retorna: una copia de la cadena convertidas las mayúsculas en minúsculas y viceversa

```
>>> cadena = "Hola Mundo"
>>> cadena.swapcase()
hOLA mUNDO
```

CONVERTIR UNA CADENA EN FORMATO TÍTULO

Método: title()
Retorna: una copia de la cadena convertida

```
>>> cadena = "hola mundo"
>>> cadena.title()
Hola Mundo
```

CENTRAR UN TEXTO

Método: center(longitud[, "caracter de relleno"])
Retorna: una copia de la cadena centrada

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.center(50, "=")
==========Bienvenido a mi aplicación===========

>>> cadena.center(50, " ")
           Bienvenido a mi aplicación
```

ALINEAR TEXTO A LA IZQUIERDA

Método: ljust(longitud[, "caracter de relleno"])
Retorna: una copia de la cadena alineada a la izquierda

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.ljust(50, "=")
Bienvenido a mi aplicación======================
```

ALINEAR TEXTO A LA DERECHA

Método: rjust(longitud[, "caracter de relleno"])
Retorna: una copia de la cadena alineada a la derecha

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.rjust(50, "=")
==========================Bienvenido a mi aplicación

>>> cadena.rjust(50, " ")
                         Bienvenido a mi aplicación
```

RELLENAR UN TEXTO ANTEPONIENDO CEROS

Método: zfill(longitud)
Retorna: una copia de la cadena rellena con ceros a la izquierda hasta alcanzar la longitud final indicada

```
>>> numero_factura = 1575
>>> str(numero_factura).zfill(12)
000000001575
```

MÉTODOS DE BÚSQUEDA

CONTAR CANTIDAD DE APARICIONES DE UNA SUBCADENA

Método: count("subcadena"[, posicion_inicio, posicion_fin])
Retorna: un entero representando la cantidad de apariciones de *subcadena* dentro de cadena

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.count("a")
3
```

BUSCAR UNA SUBCADENA DENTRO DE UNA CADENA

Método: find("subcadena"[, posicion_inicio, posicion_fin])
Retorna: un entero representando la posición donde inicia la subcadena dentro de cadena. Si no la encuentra, retorna -1

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.find("mi")
13
>>> cadena.find("mi", 0, 10)
-1
```

MÉTODOS DE VALIDACIÓN

SABER SI UNA CADENA COMIENZA CON UNA SUBCADENA DETERMINADA

Método: startswith("subcadena"[, posicion_inicio, posicion_fin])
Retorna: True o False

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.startswith("Bienvenido")
True
>>> cadena.startswith("aplicación")
False
>>> cadena.startswith("aplicación", 16)
True
```

SABER SI UNA CADENA FINALIZA CON UNA SUBCADENA DETERMINADA

Método: endswith("subcadena"[, posicion_inicio, posicion_fin])
Retorna: True o False

```
>>> cadena = "bienvenido a mi aplicación".capitalize()
>>> cadena.endswith("aplicación")
True
>>> cadena.endswith("Bienvenido")
False
>>> cadena.endswith("Bienvenido", 0, 10)
True
```

SABER SI UNA CADENA ES ALFANUMÉRICA

Método: isalnum()
Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> cadena.isalnum()
False
>>> cadena = "pepegrillo"
>>> cadena.isalnum()
True
>>> cadena = "pepegrillo75"
>>> cadena.isalnum()
True
```

SABER SI UNA CADENA ES ALFABÉTICA

Método: isalpha()
Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> cadena.isalpha()
False
```

```
>>> cadena = "pepegrillo"
>>> cadena.isalpha()
True
>>> cadena = "pepegrillo75"
>>> cadena.isalpha()
False
```

SABER SI UNA CADENA ES NUMÉRICA

Método: isdigit()
Retorna: True o False

```
>>> cadena = "pepegrillo 75"
>>> cadena.isdigit()
False
>>> cadena = "7584"
>>> cadena.isdigit()
True
>>> cadena = "75 84"
>>> cadena.isdigit()
False
>>> cadena = "75.84"
>>> cadena.isdigit()
False
```

SABER SI UNA CADENA CONTIENE SOLO MINÚSCULAS

Método: islower()
Retorna: True o False

```
>>> cadena = "pepe grillo"
>>> cadena.islower()
True
>>> cadena = "Pepe Grillo"
>>> cadena.islower()
False
>>> cadena = "Pepegrillo"
>>> cadena.islower()
False
>>> cadena = "pepegrillo75"
>>> cadena.islower()
True
```

SABER SI UNA CADENA CONTIENE SOLO MAYÚSCULAS

Método: isupper()
Retorna: True o False

```
>>> cadena = "PEPE GRILLO"
>>> cadena.isupper()
True
>>> cadena = "Pepe Grillo"
>>> cadena.isupper()
False
```

```
>>> cadena = "Pepegrillo"
>>> cadena.isupper()
False
>>> cadena = "PEPEGRILLO"
>>> cadena.isupper()
True
```

SABER SI UNA CADENA CONTIENE SOLO ESPACIOS EN BLANCO

Método: isspace()
Retorna: True o False

```
>>> cadena = "pepe grillo"
>>> cadena.isspace()
False
>>> cadena = "      "
>>> cadena.isspace()
True
```

SABER SI UNA CADENA TIENE FORMATO DE TÍTULO

Método: istitle()
Retorna: True o False

```
>>> cadena = "Pepe Grillo"
>>> cadena.istitle()
True
>>> cadena = "Pepe grillo"
>>> cadena.istitle()
False
```

MÉTODOS DE SUSTITUCIÓN

DAR FORMATO A UNA CADENA, SUSTITUYENDO TEXTO DINÁMICAMENTE

Método: format(*args, **kwargs)
Retorna: la cadena formateada

```
>>> cadena = "bienvenido a mi aplicación {0}"
>>> cadena.format("en Python")
bienvenido a mi aplicación en Python

>>> cadena = "Importe bruto: ${0} + IVA: ${1} = Importe neto: {2}"
>>> cadena.format(100, 21, 121)
Importe bruto: $100 + IVA: $21 = Importe neto: 121

>>> cadena = "Importe bruto: ${bruto} + IVA: ${iva} = Importe neto: {neto}"
>>> cadena.format(bruto=100, iva=21, neto=121)
Importe bruto: $100 + IVA: $21 = Importe neto: 121

>>> cadena.format(bruto=100, iva=100 * 21 / 100, neto=100 * 21 / 100 + 100)
Importe bruto: $100 + IVA: $21 = Importe neto: 121
```

REEMPLAZAR TEXTO EN UNA CADENA

Método: replace("subcadena a buscar", "subcadena por la cual reemplazar")
Retorna: la cadena reemplazada

```
>>> buscar = "nombre apellido"
>>> reemplazar_por = "Juan Pérez"
>>> "Estimado Sr. nombre apellido:".replace(buscar, reemplazar_por)
Estimado Sr. Juan Pérez:
```

ELIMINAR CARACTERES A LA IZQUIERDA Y DERECHA DE UNA CADENA

Método: strip(["caracter"])
Retorna: la cadena sustituida

```
>>> cadena = "  www.eugeniabahit.com  "
>>> cadena.strip()
www.eugeniabahit.com
>>> cadena.strip(' ')
www.eugeniabahit.com
```

ELIMINAR CARACTERES A LA IZQUIERDA DE UNA CADENA

Método: lstrip(["caracter"])
Retorna: la cadena sustituida

```
>>> cadena = "www.eugeniabahit.com"
>>> cadena.lstrip("w." )
eugeniabahit.com

>>> cadena = "       www.eugeniabahit.com"
>>> cadena.lstrip()
www.eugeniabahit.com
```

ELIMINAR CARACTERES A LA DERECHA DE UNA CADENA

Método: rstrip(["caracter"])
Retorna: la cadena sustituida

```
>>> cadena = "www.eugeniabahit.com     "
>>> cadena.rstrip( )
www.eugeniabahit.com
```

MÉTODOS DE UNIÓN Y DIVISIÓN

UNIR UNA CADENA DE FORMA ITERATIVA

Método: join(iterable)
Retorna: la cadena unida con el iterable (la cadena es separada por cada uno de los elementos del iterable)

```
>>> formato_numero_factura = ("Nº 0000-0", "-0000 (ID: ", ")")
```

```
>>> numero = "275"
>>> numero_factura = numero.join(formato_numero_factura)
>>> numero_factura
Nº 0000-0275-0000 (ID: 275)
```

PARTIR UNA CADENA EN TRES PARTES, UTILIZANDO UN SEPARADOR

Método: partition("separador")
Retorna: una tupla de tres elementos donde el primero es el contenido de la cadena previo al separador, el segundo, el separador mismo y el tercero, el contenido de la cadena posterior al separador

```
>>> tupla = "http://www.eugeniabahit.com".partition("www.")
>>> tupla
('http://', 'www.', 'eugeniabahit.com')

>>> protocolo, separador, dominio = tupla
>>>> "Protocolo: {0}\nDominio: {1}".format(protocolo, dominio)
Protocolo: http://
Dominio: eugeniabahit.com
```

PARTIR UNA CADENA EN VARIAS PARTES, UTILIZANDO UN SEPARADOR

Método: split("separador")
Retorna: una lista con todos elementos encontrados al dividir la cadena por un separador

```
>>> keywords = "python, guia, curso, tutorial".split(", ")
>>> keywords
['python', 'guia', 'curso', 'tutorial']
```

PARTIR UNA CADENA EN EN LÍNEAS

Método: splitlines()
Retorna: una lista donde cada elemento es una fracción de la cadena divida en líneas

```
>>> texto = """Linea 1
Linea 2
Linea 3
Linea 4
"""
>>> texto.splitlines()
['Linea 1', 'Linea 2', 'Linea 3', 'Linea 4']

>>> texto = "Linea 1\nLinea 2\nLinea 3"
>>> texto.splitlines()
['Linea 1', 'Linea 2', 'Linea 3']
```

MANIPULACIÓN DE LISTAS Y TUPLAS

En este capítulo, se verán los métodos que posee el objeto *lista*. Algunos de ellos, también se encuentran disponibles para las *tuplas*.

MÉTODOS DE AGREGADO

AGREGAR UN ELEMENTO AL FINAL DE LA LISTA

Método: append("nuevo elemento")

```
>>> nombres_masculinos = ["Alvaro", "Jacinto", "Miguel", "Edgardo", "David"]
>>> nombres_masculinos.append("Jose")
>>> nombres_masculinos
['Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose']
```

AGREGAR VARIOS ELEMENTOS AL FINAL DE LA LISTA

Método: extend(otra_lista)

```
>>> nombres_masculinos.extend(["Jose", "Gerardo"])
>>> nombres_masculinos
['Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose', 'Jose',
'Gerardo']
```

AGREGAR UN ELEMENTO EN UNA POSICIÓN DETERMINADA

Método: insert(posición, "nuevo elemento")

```
>>> nombres_masculinos.insert(0, "Ricky")
>>> nombres_masculinos
['Ricky', 'Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose',
'Jose', 'Gerardo']
```

MÉTODOS DE ELIMINACIÓN

ELIMINAR EL ÚLTIMO ELEMENTO DE LA LISTA

Método: pop()
Retorna: el elemento eliminado

```
>>> nombres_masculinos.pop()
'Gerardo'
>>> nombres_masculinos
['Ricky', 'Alvaro', 'David', 'Edgardo', 'Jacinto', 'Jose', 'Ricky', 'Jose',
'Jose']
```

ELIMINAR UN ELEMENTO POR SU ÍNDICE

Método: pop(índice)
Retorna: el elemento eliminado

```
>>> nombres_masculinos.pop(3)
'Edgardo'

>>> nombres_masculinos
['Ricky', 'Alvaro', 'David', 'Jacinto', 'Jose', 'Ricky', 'Jose', 'Jose']
```

ELIMINAR UN ELEMENTO POR SU VALOR

Método: remove("valor")

```
>>> nombres_masculinos.remove("Jose")
>>> nombres_masculinos
['Ricky', 'Alvaro', 'David', 'Jacinto', 'Ricky', 'Jose', 'Jose']
```

MÉTODOS DE ORDEN

ORDENAR UNA LISTA EN REVERSA (INVERTIR ORDEN)

Método: reverse()

```
>>> nombres_masculinos.reverse()
>>> nombres_masculinos
['Jose', 'Jose', 'Ricky', 'Jacinto', 'David', 'Alvaro', 'Ricky']
```

ORDENAR UNA LISTA EN FORMA ASCENDENTE

Método: sort()

```
>>> nombres_masculinos.sort()
>>> nombres_masculinos
['Alvaro', 'David', 'Jacinto', 'Jose', 'Jose', 'Ricky', 'Ricky']
```

ORDENAR UNA LISTA EN FORMA DESCENDENTE

Método: sort(reverse=True)

```
>>> nombres_masculinos.sort(reverse=True)
>>> nombres_masculinos
['Ricky', 'Ricky', 'Jose', 'Jose', 'Jacinto', 'David', 'Alvaro']
```

MÉTODOS DE BÚSQUEDA

CONTAR CANTIDAD DE APARICIONES ELEMENTOS

Método: count(elemento)

```
>>> nombres_masculinos = ["Alvaro", "Miguel", "Edgardo", "David", "Miguel"]
```

```
>>> nombres_masculinos.count("Miguel")
2

>>> nombres_masculinos = ("Alvaro", "Miguel", "Edgardo", "David", "Miguel")
>>> nombres_masculinos.count("Miguel")
2
```

OBTENER NÚMERO DE ÍNDICE

Método: index(elemento[, indice_inicio, indice_fin])

```
>>> nombres_masculinos.index("Miguel")
1

>>> nombres_masculinos.index("Miguel", 2, 5)
4
```

ANEXO SOBRE LISTAS Y TUPLAS

CONVERSIÓN DE TIPOS

En el conjunto de las funciones integradas de Python, es posible encontrar dos funciones que permiten convertir listas en tuplas, y viceversa. Estas funciones son *list* y *tuple*, para convertir tuplas a listas y listas a tuplas, respectivamente.

Uno de los usos más frecuentes es el de conversión de tuplas a listas, que requieran ser modificadas. Esto sucede a menudo con los resultados obtenidos a partir de una consulta a base de datos.

```
>>> tupla = (1, 2, 3, 4)
>>> tupla
(1, 2, 3, 4)

>>> list(tupla)
[1, 2, 3, 4]

>>> lista = [1, 2, 3, 4]
>>> lista
[1, 2, 3, 4]

>>> tuple(lista)
(1, 2, 3, 4)
```

CONCATENACIÓN DE COLECCIONES

Se pueden concatenar (o unir) dos o más listas o dos o más tuplas, mediante el signo de adición +.

No puede unirse una lista a una tupla. Las colecciones a unir deben ser del mismo tipo.

```
>>> lista1 = [1, 2, 3, 4]
>>> lista2 = [3, 4, 5, 6, 7, 8]
>>> lista3 = lista1 + lista2
>>> lista3
[1, 2, 3, 4, 3, 4, 5, 6, 7, 8]

>>> tupla1 = (1, 2, 3, 4, 5)
>>> tupla2 = (4, 6, 8, 10)
>>> tupla3 = (3, 5, 7, 9)
>>> tupla4 = tupla1 + tupla2 + tupla3
>>> tupla4
(1, 2, 3, 4, 5, 4, 6, 8, 10, 3, 5, 7, 9)
```

VALOR MÁXIMO Y MÍNIMO

Se puede obtener el valor máximo y mínimo tanto de listas como de tuplas:

```
>>> max(tupla4)
10
>>> max(tupla1)
5
>>> min(tupla1)
1
>>> max(lista3)
8
>>> min(lista1)
1
```

CONTAR ELEMENTOS

La función `len()` sirve tanto para contar elementos de una lista o tupla, como caracteres de una cadena de texto:

```
>>> len(lista3)
10
>>> len(lista1)
4
```

MANIPULACIÓN DE DICCIONARIOS

MÉTODOS DE ELIMINACIÓN

VACIAR UN DICCIONARIO

Método: clear()

```
>>> diccionario = {"color": "violeta", "talle": "XS", "precio": 174.25}
>>> diccionario
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> diccionario.clear()
>>> diccionario
{}
```

MÉTODOS DE AGREGADO Y CREACIÓN

COPIAR UN DICCIONARIO

Método: copy()

```
>>> diccionario = {"color": "violeta", "talle": "XS", "precio": 174.25}
>>> remera = diccionario.copy()
>>> diccionario
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> remera
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> diccionario.clear()
>>> diccionario
{}

>>> remera
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> musculosa = remera
>>> remera
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> musculosa
{'color': 'violeta', 'precio': 174.25, 'talle': 'XS'}

>>> remera.clear()
>>> remera
{}

>>> musculosa
{}
```

CREAR UN NUEVO DICCIONARIO DESDE LAS CLAVES DE UNA SECUENCIA

Método: dict.fromkeys(secuencia[, valor por defecto])

```
>>> secuencia = ["color", "talle", "marca"]
>>> diccionario1 = dict.fromkeys(secuencia)
>>> diccionario1
{'color': None, 'marca': None, 'talle': None}

>>> diccionario2 = dict.fromkeys(secuencia, 'valor x defecto')
>>> diccionario2
{'color': 'valor x defecto', 'marca': 'valor x defecto', 'talle': 'valor x
defecto'}
```

CONCATENAR DICCIONARIOS

Método: update(diccionario)

```
>>> diccionario1 = {"color": "verde", "precio": 45}
>>> diccionario2 = {"talle": "M", "marca": "Lacoste"}
>>> diccionario1.update(diccionario2)
>>> diccionario1
{'color': 'verde', 'precio': 45, 'marca': 'Lacoste', 'talle': 'M'}
```

ESTABLECER UNA CLAVE Y VALOR POR DEFECTO

Método: setdefault("clave"[, None|valor_por_defecto])

> Si la clave no existe, la crea con el valor por defecto. Siempre retorna el valor para la clave pasada como parámetro.

```
>>> remera = {"color": "rosa", "marca": "Zara"}
>>> clave = remera.setdefault("talle", "U")
>>> clave
'U'

>>> remera
{'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}

>>> remera2 = remera.copy()
>>> remera2
{'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}

>>> clave = remera2.setdefault("estampado")
>>> clave
>>> remera2
{'color': 'rosa', 'estampado': None, 'marca': 'Zara', 'talle': 'U'}

>>> clave = remera2.setdefault("marca", "Lacoste")
>>> clave
'Zara'
```

```
>>> remera2
{'color': 'rosa', 'estampado': None, 'marca': 'Zara', 'talle': 'U'}
```

MÉTODOS DE RETORNO

OBTENER EL VALOR DE UNA CLAVE

Método: get(clave[, "valor x defecto si la clave no existe"])

```
>>> remera.get("color")
'rosa'

>>> remera.get("stock")
>>> remera.get("stock", "sin stock")
'sin stock'
```

SABER SI UNA CLAVE EXISTE EN EL DICCIONARIO

Método: 'clave' in diccionario

```
>>> existe = 'precio' in remera
>>> existe
False

>>> existe = 'color' in remera
>>> existe
True
```

OBTENER LAS CLAVES Y VALORES DE UN DICCIONARIO

Método: items()

```
diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}

for clave, valor in diccionario.items():
    clave, valor
```

Salida:

```
('color', 'rosa')
('marca', 'Zara')
('talle', 'U')
```

OBTENER LAS CLAVES DE UN DICCIONARIO

Método: keys()

```
diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}
for clave in diccionario.keys():
    clave
'marca'
'talle'
'color'
```

Obtener claves en una lista

```
>>> diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}
>>> claves = list(diccionario.keys())
>>> claves
['color', 'marca', 'talle']
```

OBTENER LOS VALORES DE UN DICCIONARIO

Método: values()

```
diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}
for clave in diccionario.values():
    clave
'rosa'
'Zara'
'U'
```

Obtener valores en una lista

```
>>> diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}
>>> claves = list(diccionario.values())
```

OBTENER LA CANTIDAD DE ELEMENTOS DE UN DICCIONARIO

Para contar los elementos de un diccionario, al igual que con las listas y tuplas, se utiliza la función integrada `len()`

```
>>> diccionario = {'color': 'rosa', 'marca': 'Zara', 'talle': 'U'}
>>> len(diccionario)
3
```

MANEJO Y MANIPULACIÓN DE ARCHIVOS

Python permite trabajar en dos niveles diferentes con respecto al sistema de archivos y directorios.

Uno de ellos, es a través del módulo *os*, que facilita el trabajo con todo el sistema de archivos y directorios, a nivel del propios Sistema Operativo.

El segundo nivel, es el que permite trabajar con archivos manipulando su lectura y escritura desde la propia aplicación o *script*, tratando a cada archivo como un objeto.

MODOS DE APERTURA DE UN ARCHIVO

El **modo de apertura de un archivo**, está relacionado con el objetivo final que responde a la pregunta ***"¿para qué se está abriendo este archivo?"***. Las respuestas pueden ser varias: para leer, para escribir, o para leer y escribir.

Cada vez que se "abre" un archivo se está creando un ***puntero en memoria***.

Este puntero posicionará un ***cursor*** (o *punto de acceso*) en un lugar específico de la memoria (dicho de modo más simple, posicionará el cursor en un *byte* determinado del contenido del archivo).

Este cursor se moverá dentro del archivo, a medida que se lea o escriba en dicho archivo.

Cuando un archivo se abre en modo lectura, el cursor se posiciona en el *byte 0* del archivo (es decir, al comienzo del archivo). Una vez leído el archivo, el cursor pasa al *byte final* del archivo (equivalente a cantidad total de *bytes* del archivo). Lo mismo sucede cuando se abre en modo escritura. El cursor se moverá a medida que se va escribiendo.

Cuando se desea escribir al final de un archivo no nulo, se utiliza el modo *append* (agregar). De esta forma, el archivo se abre con el cursor al final del archivo.

El símbolo + como sufijo de un modo, agrega el modo contrario al de apertura una vez se ejecute la acción de apertura. Por ejemplo, el modo r (read) con el sufijo + (r+), abre el archivo para lectura, y tras la lectura, vuelve el cursor al *byte* 0.

La siguiente tabla muestra los diferentes modos de apertura de un archivo:

Indicador	Modo de apertura	Ubicación del puntero
`r`	Solo lectura	Al inicio del archivo
`rb`	Solo lectura en modo binario	Al inicio del archivo
`r+`	Lectura y escritura	Al inicio del archivo
`rb+`	Lectura y escritura en modo binario	Al inicio del archivo
`w`	Solo escritura. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
`wb`	Solo escritura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
`w+`	Escritura y lectura. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
`wb+`	Escritura y lectura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
`a`	Añadido (agregar contenido). Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.
`ab`	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.
`a+`	Añadido (agregar contenido) y lectura.	Si el archivo existe, al final de

	Crea el archivo si éste no existe.	éste. Si el archivo no existe, al comienzo.
ab+	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.

ALGUNOS MÉTODOS DEL OBJETO FILE

El objeto file, entre sus métodos dispone de los siguientes:

Método	Descripción
`read([bytes])`	Lee todo el contenido de un archivo. Si se le pasa la longitud de bytes, leerá solo el contenido hasta la longitud indicada.
`readlines()`	Lee todas las líneas de un archivo
`write(cadena)`	Escribe *cadena* dentro del archivo
`writelines(secuencia)`	Secuencia será cualquier iterable cuyos elementos serán escritos uno por línea

ACCESO A ARCHIVOS MEDIANTE LA ESTRUCTURA WITH

Con la estructura with y la función open(), puede abrirse un archivo en cualquier modo y trabajar con él, sin necesidad de cerrarlo o destruir el puntero, ya que de esto se encarga la estructura with.

Leer un archivo:

```
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
```

Escribir en un archivo:

```
contenido = """
    Este será el contenido del nuevo archivo.
    El archivo tendrá varias líneas.
```

```
"""

with open("archivo.txt", "r") as archivo:
    archivo.write(contenido)
```

MANEJO DE ARCHIVOS CSV

El formato **CSV** deriva su nombre del inglés «*comma separated values*» (valores separados por coma), definido en las RFC 4180. Se trata de archivos de texto plano, destinados al almacenamiento masivo de datos. Es uno de los formatos más simples para efectuar análisis de datos. De hecho, muchos formatos de archivo no libres (o libres pero más complejos), suelen pasarse a formato CSV para aplicar ciencia de datos compleja con diversos lenguajes.

Un archivo CSV se encuentra formado por una cabecera que define nombres de columnas, y las filas siguientes, tienen los datos correspondientes a cada columna, separados por una coma. Sin embargo, muchos otros símbolos pueden utilizarse como separadores de celdas. Entre ellos, el tabulado y el punto y coma son igual de frecuentes que la coma.

ALGUNOS EJEMPLOS DE ARCHIVOS CSV

Datos meteorológicos (separados por ;)

```
ID;DATA;VV;DV;T;HR;PPT;RS;P
0;2016-03-01 00:00:00;;;9.9;73;;;
1;2016-03-01 00:30:00;;;9.0;67;;;
2;2016-03-01 01:00:00;;;8.3;64;;;
3;2016-03-01 01:30:00;;;8.0;61;;;
4;2016-03-01 02:00:00;;;7.4;62;;;
5;2016-03-01 02:30:00;;;8.3;47;;;
6;2016-03-01 03:00:00;;;7.7;50;;;
7;2016-03-01 03:30:00;;;9.0;39;;;
```

Puntajes obtenidos por jugadores de un torneo (separados por ,)

```
nombre,cantidad,anio
```

```
Maria,858,1930
Jose,665,1930
Rosa,591,1930
Juan Carlos,522,1930
Antonio,509,1930
Maria Esther,495,1930
Maria Luisa,470,1930
Juana,453,1930
Juan,436,1930
```

Empresas registradas en la Inspección General de Justicia de Argentina (separados por , y datos entrecomillados)

```
"numero_correlativo","tipo_societario","descripcion_tipo_societario","razon_soci
al","dada_de_baja","codigo_baja","detalle_baja"
"10","10","SOCIEDAD COLECTIVA","A A VALLE Y COMPA¥IA","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
"11","10","SOCIEDAD COLECTIVA","A LUCERO Y H CARATOLI","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
"12","10","SOCIEDAD COLECTIVA","A PUIG E HIJOS","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
"13","10","SOCIEDAD COLECTIVA","A C I C A","S","42014","PERTENECE A REGISTRO
ENTIDADES INACTIVAS"
"14","10","SOCIEDAD COLECTIVA","A¥ON BEATRIZ S Y CIA","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
"15","10","SOCIEDAD COLECTIVA","ABA DIESEL","S","42014","PERTENECE A REGISTRO
ENTIDADES INACTIVAS"
"16","10","SOCIEDAD COLECTIVA","ABADA L JOSE Y JORGE JOSE
ABADAL","S","42014","PERTENECE A REGISTRO ENTIDADES INACTIVAS"
"17","10","SOCIEDAD COLECTIVA","ABADAL JOSE E HIJO","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
"18","10","SOCIEDAD COLECTIVA","ABATE Y MACIAS","S","42014","PERTENECE A
REGISTRO ENTIDADES INACTIVAS"
```

Es posible también, encontrar datos almacenados en archivos de texto (TXT) con formatos muy similares al que se espera encontrar en un CSV. A veces es posible desarrollar un script de formato para corregir estos archivos y así poder trabajar con un CSV.

Observaciones meteorológicas en TXT

```
FECHA    TMAX  TMIN  NOMBRE
-------- ----- ----- ----------------------------------------
07122017  28.0  19.0 AEROPARQUE AERO
07122017  26.8  12.4 AZUL AERO
```

```
07122017  29.6   7.8 BAHIA BLANCA AERO
07122017  22.7   6.7 BARILOCHE AERO
07122017   3.0  -8.5 BASE BELGRANO II
07122017   2.4  -0.2 BASE CARLINI (EX JUBANY)
07122017   3.9  -0.6 BASE ESPERANZA
07122017   0.7  -3.6 BASE MARAMBIO
```

TRABAJAR CON ARCHIVOS CSV DESDE PYTHON

Python provee de un módulo propio llamado csv, que facilita el parseo de los datos de archivos CSV, tanto para lectura como escritura.

Este módulo, se utiliza en combinación con la estructura with y la función open, para leer o generar el archivo, y el módulo CSV para su análisis (*parsing*).

LECTURA DE ARCHIVOS CSV

Contenido de archivo.csv

```
0;2016-03-01 00:00:00;;;9.9;73;;;
1;2016-03-01 00:30:00;;;9.0;67;;;
2;2016-03-01 01:00:00;;;8.3;64;;;
3;2016-03-01 01:30:00;;;8.0;61;;;
4;2016-03-01 02:00:00;;;7.4;62;;;
5;2016-03-01 02:30:00;;;8.3;47;;;
6;2016-03-01 03:00:00;;;7.7;50;;;
7;2016-03-01 03:30:00;;;9.0;39;;;
8;2016-03-01 04:00:00;;;8.7;39;;;
```

```
from csv import reader

with open("archivo.csv", "r") as archivo:
    documento = reader(archivo, delimiter=';', quotechar='"')
    for fila in documento:
        ' '.join(fila)
```

Salida:

```
'0 2016-03-01 00:00:00   9.9 73   '
'1 2016-03-01 00:30:00   9.0 67   '
'2 2016-03-01 01:00:00   8.3 64   '
'3 2016-03-01 01:30:00   8.0 61   '
'4 2016-03-01 02:00:00   7.4 62   '
'5 2016-03-01 02:30:00   8.3 47   '
'6 2016-03-01 03:00:00   7.7 50   '
'7 2016-03-01 03:30:00   9.0 39   '
'8 2016-03-01 04:00:00   8.7 39   '
```

Cuando el archivo CSV tiene una cabecera, es necesario saltar dicho encabezado:

Contenido de archivo.csv

```
ID;DATA;VV;DV;T;HR;PPT;RS;P
0;2016-03-01 00:00:00;;;9.9;73;;;
1;2016-03-01 00:30:00;;;9.0;67;;;
2;2016-03-01 01:00:00;;;8.3;64;;;
3;2016-03-01 01:30:00;;;8.0;61;;;
4;2016-03-01 02:00:00;;;7.4;62;;;
5;2016-03-01 02:30:00;;;8.3;47;;;
6;2016-03-01 03:00:00;;;7.7;50;;;
7;2016-03-01 03:30:00;;;9.0;39;;;
8;2016-03-01 04:00:00;;;8.7;39;;;
```

```
from csv import reader


with open("archivo.csv", "r") as archivo:
    documento = reader(archivo, delimiter=';', quotechar='"')
    cabeceras = next(documento)
    for fila in documento:
        ' '.join(fila)
```

Salida:

```
'0 2016-03-01 00:00:00   9.9 73   '
'1 2016-03-01 00:30:00   9.0 67   '
'2 2016-03-01 01:00:00   8.3 64   '
'3 2016-03-01 01:30:00   8.0 61   '
'4 2016-03-01 02:00:00   7.4 62   '
'5 2016-03-01 02:30:00   8.3 47   '
'6 2016-03-01 03:00:00   7.7 50   '
'7 2016-03-01 03:30:00   9.0 39   '
'8 2016-03-01 04:00:00   8.7 39   '
```

Otra forma de leer archivos CSV con cabeceras, es utilizar el objeto ***DictReader*** en vez de *reader*, y así acceder solo al valor de las columnas deseadas, por su nombre:

```
from csv import DictReader


with open("archivo.csv", "r") as archivo:
    documento = DictReader(archivo, delimiter=';', quotechar='"')
    for fila in documento:
        fila['DATA']
```

Salida:

```
'2016-03-01 00:00:00'
'2016-03-01 00:30:00'
'2016-03-01 01:00:00'
'2016-03-01 01:30:00'
'2016-03-01 02:00:00'
'2016-03-01 02:30:00'
'2016-03-01 03:00:00'
'2016-03-01 03:30:00'
'2016-03-01 04:00:00'
```

ESCRITURA DE ARCHIVOS CSV

Escritura de un CSV sin cabecera:

```
from csv import writer


with open("datos.csv", "w") as archivo:
    documento = writer(archivo, delimiter=';', quotechar='"')
    documento.writerows(matriz)
```

En el ejemplo anterior, una matriz podría ser una lista de listas con igual cantidad de elementos. Por ejemplo:

```
matriz = [
    ['Juan', 373, 1970],
    ['Ana', 124, 1983],
    ['Pedro', 901, 1650],
    ['Rosa', 300, 2000],
    ['Juana', 75, 1975],
]
```

Lo anterior, generaría un archivo llamado *datos.csv* con el siguiente contenido:

```
eugenia@bella:~$ cat datos.csv
Juan;373;1970
Ana;124;1983
Pedro;901;1650
Rosa;300;2000
Juana;75;1975
```

Escritura de un CSV con cabecera:

En este caso, la matriz a ser escrita requerirá ser una lista de diccionarios cuyas claves coincidan con las cabeceras indicadas.

```
matriz = [
    dict(jugador='Juan', puntos=373, anio=1970),
    dict(jugador='Ana', puntos=124, anio=1983),
    dict(jugador='Pedro', puntos=901, anio=1650),
    dict(jugador='Rosa', puntos=300, anio=2000),
    dict(jugador='Juana', puntos=75, anio=1975),
]


from csv import DictWriter

cabeceras = ['jugador', 'puntos', 'anio']


with open("datos.csv", "w") as archivo:
    documento = DictWriter(archivo, delimiter=';', quotechar='"',
       fieldnames=cabeceras)
    documento.writeheader()
    documento.writerows(matriz)
```

Funciones estadísticas simples

Sobre listas y tuplas obtenidas o no a partir de un CSV, pueden efectuarse funciones estadísticas simples como las siguientes:

```
Contar elementos        len(coleccion)
Sumar elementos         sum(coleccion)
Obtener número mayor    max(coleccion)
Obtener numero menor    min(coleccion)
```

PROBABILIDAD Y ESTADÍSTICA CON PYTHON

PROBABILIDAD DE SUCESOS SIMPLES Y COMPUESTOS MUTUAMENTE EXCLUYENTES EN PYTHON

ESPACIO MUESTRAL

Un **espacio muestral** es un conjunto de sucesos posibles, como los que podrían resultar al lanzar un dado:

$$E = \{1, 2, 3, 4, 5, 6\}$$

```
espacio_muestral = [1, 2, 3, 4, 5, 6]
```

Se refiere como **punto muestral** a cada elemento en un espacio muestral. La cantidad de puntos muestrales se denota por n tal que para el espacio muestral $E = \{1, 2, 3, 4, 5, 6\}$, $n = 6$.

```
n = len(espacio_muestral)
```

SUCESOS SIMPLES Y COMPUESTOS

Un **suceso**, es un conjunto de resultados dentro de un espacio muestral. Por ejemplo:

- el lanzamiento de un dado es un suceso
- la probabilidad de que en dicho lanzamiento salga el número 5, es un **suceso simple** $A = \{5\}$ y es **excluyente**: si sale 5, no puede simultáneamente salir ningún otro número.
- la probabilidad de que en el lanzamiento salga un número impar, es el **suceso compuesto** $B = \{1, 3, 5\}$ que dependerá a su vez de los sucesos simples excluyentes $B_1 = \{1\}$, $B_2 = \{2\}$ y $B_3 = \{3\}$

ASIGNACIÓN DE PROBABILIDADES

La asignación de probabilidades es aquella que provee modelos matemáticos para calcular las posibilidades de que sucesos específicos ocurran o no.

La probabilidad de un suceso se denota por $P(suceso)$.

Los sucesos pueden ser:

- simples o compuestos
- mutuamente excluyentes o independientes

SUCESOS SIMPLES MUTUAMENTE EXCLUYENTES

Si se considera un espacio muestral A , cada uno de los puntos muestrales k , quedará denotado por A_k y la probabilidad de éstos, designada como $P(A_k)$, quedará determinada por:

$$P(A_k) = \frac{1}{n}$$

```
probabilidad = 1.0 / n
```

> En **Python**, se requiere que al menos un elemento de la ecuación sea un número real si lo que se requiere como resultado es un número real.

La probabilidad de cada punto muestral, como sucesos excluyentes entre sí, es la misma para cada suceso.

$$P(6) = P(5) = P(4) = P(3) = P(2) = P(1) = \frac{1}{n} = \frac{1}{6}$$

SUCESOS COMPUESTOS POR SUCESOS SIMPLES MUTUAMENTE EXCLUYENTES

Cuando los sucesos simples que conforma al suceso compuesto A son mutuamente excluyente, la probabilidad del suceso compuesto estará dada por la suma de las probabilidades de cada suceso simple $P(A_k)$, tal que:

$$P(A) = P(A_1) + P(A_2) + ... + P(A_k)$$

Por ejemplo, para estimar la probabilidad de que en un único lanzamiento de dado, salga un número par, se obtiene el suceso $A = \{2, 4, 6\}$, dado por suma de las probabilidades de cada uno de sucesos simples $P(2) + P(3) + P(4)$ del espacio muestral $E = \{1, 2, 3, 4, 5, 6\}$ tal que:

$$P(A) = P(2) + P(4) + P(6)$$
$$P(A) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{3}{6}$$
$$P(A) = \frac{1}{2}$$

En el primer resultado $\frac{3}{6}$ (en el segundo paso, antes de hallar el máximo común divisor [MCD] y reducir la fracción a $\frac{1}{2}$), el denominador es equivalente a la cantidad de sucesos simples dentro del suceso compuesto «números pares» y se denota por h . El denominador, 6 , es n , el total de todos los sucesos del espacio muestral. De esta forma, la probabilidad de un suceso compuesto A por sucesos mutuamente excluyentes queda dada por el cociente de h y n tal que:

$$P(A) = \frac{h}{n}$$

```
numeros_pares = [i for i in espacio_muestral if i % 2 is 0]
h = len(numeros_pares)
probabilidad = float(h) / n
```

Un suceso compuesto se puede denotar por la unión de sus sucesos simples (símbolo $\cup$, leído como "o"), tal que:

$$P(A_1 \cup A_2 \cup ...A_k) = P(A_1) + P(A_2) + ...A_k$$

Por ejemplo, para el caso del suceso «números pares», se obtiene que:

$$P(2 \cup 4 \cup 6) = P(2) + P(4) + P(6)$$
$$P(2 \cup 4 \cup 6) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{3}{6}$$
$$P(2 \cup 4 \cup 6) = \frac{1}{2}$$

Tal que $P(2 \cup 4 \cup 6)$ es un suceso y $P(2)$, $P(4)$ y $P(6)$ son las probabilidades de los tres sucesos que lo componen. En un nuevo contexto, $P(2 \cup 4 \cup 6)$ puede ser tratado como un suceso A .

FUNCIONES

```
# Probabilidad de sucesos simples mutuamente excluyentes
pssme = lambda e: 1.0 / len(e)


# Probabilidad de sucesos compuestos mutuamente excluyentes
def pscme(e, sc):
    n = len(e)
    return len(sc) / float(n)
```

PROBABILIDAD CONDICIONAL EN PYTHON

c. Probabilidad de B: $B = \{2, 4, 6\}$ $P(B) = \frac{h}{n} = \frac{3}{6} = \frac{1}{2}$

d. Probabilidad de la intersección:

$$P(A \cap B) = P(A)P(B)$$
$$P(A \cap B) = \frac{1}{2} \cdot \frac{1}{2}$$
$$P(A \cap B) = \frac{1}{4}$$

```
e = espacio_muestral = [1, 2, 3, 4, 5, 6]
n = len(e)  # total de la muestra

# probabilidad de A
a = [i for i in e if i % 2 is not 0]
pa = len(a) / float(n)
```

```
# probabilidad de B
b = [i for i in e if i % 2 is 0]
pb = len(b) / float(n)

# probabilidad de la intersección de sucesos
pi = pa * pb
```

FUNCIONES

```
# Probabilidad condicional: sucesos dependientes
def pscd(e, a, b):
    i = list(set(a).intersection(b))
    pi = pscme(e, i)
    pa = pscme(e, a)
    return pi / pa


# Probabilidad condicional: sucesos independientes
def psci(e, a, b):
    pa = pscme(e, a)
    pb = pscme(e, b)
    return pa * pb
```

SUCESOS DEPENDIENTES

Se refiere a la probabilidad de que dos sucesos ocurran simultáneamente siendo que el segundo suceso depende de la ocurrencia del primero.

La probabilidad de que ocurra B si ocurre A, se denota por $P(B|A)$ y se lee como *"la probabilidad de B dado A"*, tal que:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Donde $P(A \cap B)$ es la probabilidad de la intersección de los sucesos de A y B — definida como: $P(A \cap B) \equiv P(A)P(B|A)$ —, tal que la intersección es un nuevo suceso compuesto por sucesos simples. En el siguiente ejemplo, equivaldría a $\{1, 3\}$ (porque 1 y 3 están tanto en A como en B).

Ejemplo: ¿qué probabilidad existe de que al lanzar un dado resulte un número impar menor que 4?

El lanzamiento del dado es un suceso en sí mismo. Se desea averiguar la probabilidad de $B = \{1, 2, 3\}$ (número menor que 4) dado que $A = \{1, 3, 5\}$ (número impar) ocurriese en el espacio muestral $E = \{1, 2, 3, 4, 5, 6\}$.

```
espacio_muestral = [1, 2, 3, 4, 5, 6]
a = [i for i in espacio_muestral if i % 2 is not 0]
b = [i for i in espacio_muestral if i < 4]
```

Para calcular la probabilidad de una intersección, primero se obtiene la intersección:

$$A \cap B = \{1, 3\}$$

```
intersec = [i for i in a if i in b]
```

Y luego, se calcula la probabilidad del nuevo suceso compuesto $\{1, 3\}$:

$$P(A \cap B) = P(1) + P(3) = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = \frac{1}{3}$$

o, lo que es igual:

$$P(A \cap B) = \frac{h}{n} = \frac{2}{6} = \frac{1}{3}$$

Es necesario además, obtener la probabilidad de A, teniendo en cuenta que es también un suceso compuesto:

$$P(A) = \frac{h}{n} = \frac{3}{6} = \frac{1}{2}$$

Finalmente, se obtiene que:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$
$$P(B|A) = \frac{1/3}{1/2}$$
$$P(B|A) = \frac{2}{3} = 0.\bar{6}$$

```
e = espacio_muestral = [1, 2, 3, 4, 5, 6]

a = [i for i in e if i % 2 is not 0]  # números impares
b = [i for i in e if i < 4]           # números menores que 4
```

```
intersec = [i for i in a if i in b]   # intersección de A y B

n = len(e)                  # total de la muestra
ha = len(a)                 # total de sucesos simples en A
hintersec = len(intersec)  # total de sucesos simples en la intersección

# probabilidad de la intersección
probabilidad_intersec = float(hintersec) / n

# probabilidad de 'a'
probabilidad_a = float(ha) / n

# probabilidad condicional
probabilidad_b_dado_a = probabilidad_intersec / probabilidad_a
```

TEORÍA DE CONJUNTOS EN PYTHON

Al obtener la intersección de dos sucesos compuestos se ha empleado un método manual al decir: devolver 'i' por cada 'i' en la lista 'a' si está en la lista 'b'.

No obstante, dado que cada suceso compuesto es un conjunto y que Python provee un tipo de datos llamado `set` (conjunto), es posible obtener la intersección manipulando los sucesos compuestos como conjuntos de Python. Con `set` se puede convertir cualquier iterable a conjunto y realizar operaciones de conjuntos como unión e intersección cuando sea necesario.

```
intersec = list(set(a).intersection(b))
```

Aquí el conjunto obtenido se convierte en lista a fin a fin de guardar coherencia con el resto del código y que el elemento resultante soporte las operaciones y tratamiento habituales de una lista. Ante la duda de si corresponde usar listas o conjuntos, se debe aplicar el principio de simplicidad e implementar la solución más simple.

SUCESOS INDEPENDIENTES

A diferencia del caso anterior, aquí la probabilidad de que ocurra B no está afecta por la ocurrencia de A . Por ejemplo, la probabilidad de lanzar un dado

y obtener un número par (suceso B) no está afectada por el hecho de que en un lanzamiento previo se obtuviese un número impar (suceso A). La probabilidad de B es independiente de A y está dada por el producto de la probabilidad de ambos sucesos:

$$P(A \cap B) = P(A)P(B)$$

Aquí la intersección es la probabilidad de que confluyan ambos sucesos.

Calculada la probabilidad de ambos sucesos independientes, se multiplican obteniendo:

a. Espacio muestral (para ambos sucesos):

$$E = \{1, 2, 3, 4, 5, 6\}$$

b. Probabilidad de A:

$$A = \{1, 3, 5\}$$
$$P(A) = \frac{h}{n} = \frac{3}{6} = \frac{1}{2}$$

TEOREMA DE BAYES EN PYTHON

TEOREMA DE BAYES Y PROBABILIDAD DE CAUSAS

Dada una serie de sucesos A_k cuya suma total es un espacio muestral E y un suceso B cualquiera, el Teorema de Bayes permite conocer la probabilidad de que cada suceso A_k de E , sea la causa de B . Por este motivo, también se lo conoce como **probabilidad de causas**.

DATOS: CASO PRÁCTICO

Dada una ciudad de **50 000 habitantes**, con la siguiente distribución:

Niñas	Niños	Mujeres	Hombres
11000	9000	16000	14000

Y, un reporte de **9 000 casos de gripe**, distribuidos de la siguiente forma:

Niñas	Niños	Mujeres	Hombres
2000	1500	3000	2500

Se pretende obtener la probabilidad de que la causa de contraer gripe sea el hecho de pertences a un determinado sector demográfico (por ejemplo, al sector demográfico conformado por niños o niñas).

ANÁLISIS

De lo expuesto *ut supra* se obtiene que:

- La ciudad (total absoluto de habitantes) es el espacio muestral E .
- La cantidad de niñas, niños, mujeres y hombres es cada uno de los sucesos A_k del espacio muestral E
- Como valor de n se toma la suma del espacio muestral $\sum A_k$, tal que $n = 50000$
- El valor de h para los sucesos A_k es cada uno de los valores dados en la tabla de distribución de habitantes.
- Tener gripe es el suceso B .
- La tabla de distribución de casos de gripe, se corresponde a las intersecciones del suceso B con cada suceso A_k , es decir a cada $A_k \cap B$

Según el cálculo de probabilidad que se aplique, se podrá obtener:

- La probabilidad de ser niña, niño, mujer u hombre en la ciudad, por medio de $P(A_k)$. Se considera una **probabilidad a priori**.
- La probabilidad de ser niña, niño, mujer u hombre y tener gripe, que se obtiene con $P(A_k|\ B)$ y se considera una **probabilidad condicional**.
- La probabilidad de que cualquier habitante, independientemente del sector al que pertenezca tenga gripe, se obtiene con

$$P(B) = \sum_{k=1}^{n} P(A_k)\ P(B|A_k)$$

y se la considera una **probabilidad total.**

- La probabilidad de que alguien con gripe sea niña, niño, mujer u hombre se obtiene con el **Teorema de Bayes**. A esta probabilidad se la considera una **probabilidad a posteriori**, permitiendo dar respuesta a preguntas cómo ¿cuál es la probabilidad de que un nuevo caso de gripe sea de en un niño o niña?

Una forma eficaz y ordenada de obtener una probabilidad a *posteriori* con el Teorema de Bayes, es obtener primero las tres probabilidades previas: a priori, condicional y total.

> **AVISO:**
> en lo sucesivo, se utilizará `map(float, <lista>)` en el código fuente, para convertir los elementos de una lista en números reales, toda vez que hacerlo no sobrecargue el código.

PROCEDIMIENTO

1. Cálculo de probabilidad *a priori*

Retorna: probabilidad de que un habitante pertenezca a un sector demográfico específico.

Fórmula: $P(A_k) = \frac{h}{n}$

Datos necesarios:

h_k = datos de la tabla de distribución de habitantes

n = siempre es la cantidad total del espacio muestral (50 000)

Resultados:

$P(A_1) = \frac{11000}{50000} = 0.22$ probabilidad de ser niña

$P(A_2) = \frac{9000}{50000} = 0.18$ probabilidad de ser niño

$P(A_3) = \frac{16000}{50000} = 0.32$ probabilidad de ser mujer

$P(A_4) = \frac{14000}{50000} = 0.28$ probabilidad de ser hombre

Código Python:

```
habitantes = map(float, [11000, 9000, 16000, 14000])
n = sum(habitantes)
pa = [h / n for h in habitantes]
```

2. Probabilidad condicional

Retorna: probabilidad de tener gripe perteneciendo a un sector demográfico específico.

Certeza: A_k (el sector demográfico)

Objetivo: B (la probabilidad de tener gripe)

Fórmula: $$P(B|\ A_k) = \frac{P(A_k \cap\ B)}{P(A_k)}$$

Datos necesarios:

$$P(A_k \cap\ B) = \frac{h = B_k}{n = 50000}$$

h = intersecciones (datos de la tabla de distribución de casos de gripe)

Resultados:

$P(B|\ A_1) = \frac{\frac{2000}{50000}}{0.22} = 0.\overline{18}$ probabilidad de tener gripe siendo niña

$P(B|\ A_2) = \frac{\frac{1500}{50000}}{0.18} = 0.1\bar{6}$ probabilidad de tener gripe siendo niño

$P(B|\ A_3) = \frac{\frac{3000}{50000}}{0.32} = 0.19$ probabilidad de tener gripe siendo mujer

$P(B|\ A_4) = \frac{\frac{2500}{50000}}{0.28} = 0.18$ probabilidad de tener gripe siendo hombre

Código Python:

```
afectados = map(float, [2000, 1500, 3000, 2500])
pi = [k / n for k in afectados]
pba = [pi[i] / pa[i] for i in range(len(pi))]
```

3. Probabilidad total

Retorna: probabilidad de que cualquiera de los habitantes, independientemente del sector demográfico al que pertenezcan, pueda tener gripe.

Fórmula:

$$P(B) = \sum_{k=1}^{n} P(A_k)\, P(B|A_k)$$

Datos necesarios:

probabilidad a priori

probabilidad condicional

Resultados:

$$P(B) = P(A_1)P(B|A_1)+P(A_2)P(B|A_2)+P(A_3)P(B|A_3)+P(A_4)P(B|A_4)$$

$$P(B) = 0.22 \cdot 0.\overline{18} + 0.18 \cdot 0.1\bar{6} + 0.32 \cdot 0.19 + 0.28 \cdot 0.18$$

$$P(B) = 0.04 + 0.03 + 0.06 + 0.05$$

$$P(B) = 0.18$$

Código Python:

```
productos = [pa[i] * pba[i] for i in range(len(pa))]
pb = sum(productos)
```

Observaciones:

(a) notar que en la salida anterior existirá una diferencia de .01 con respecto a la solución manual. Esto es debido al redondeo efectuado en la solución manual. Dicha diferencia puede ser erradicada empleando 3 decimales en los valores de la probabilidad condicional (en lugar de dos) en la solución manual.

(b) la probabilidad de NO tener gripe estará dada por $1 - P(B)$ tal que $1 - 0.18 = 0.82$ pero no será necesario utilizarla para este ejemplo con el Teorema de Bayes.

4. Probabilidad a *posteriori*

Retorna: probabilidad de pertenecer a un sector demográfico específico y tener gripe.

Certeza: B (tener gripe)

Objetivo: A_k (la probabilidad de pertenecer a un sector demográfico concreto)

Fórmula:

$$P(A_k|\ B) = \frac{P(A_k)\ P(B|A_k)}{\sum_{k=1}^{n} P(A_k)\ P(B|A_k)}$$

Datos necesarios:

$P(A_k)\ P(B|A_k)$ = el producto obtenido en cada uno de los términos de la probabilidad total

$\sum_{k=1}^{n} P(A_k)\ P(B|A_k)$ = la probabilidad total

Resultados:

$P(A_1|\ B) = \frac{0.04}{0.18} = 0.\overline{22}$ probabilidad de ser niña teniendo gripe

$P(A_2|\ B) = \frac{0.03}{0.18} = 0.1\bar{6}$ probabilidad de ser niño teniendo gripe

$P(A_3|\,B) = \frac{0.06}{0.18} = 0.\overline{33}$ probabilidad de ser mujer teniendo gripe

$P(A_4|\,B) = \frac{0.05}{0.18} = 0.2\overline{7}$ probabilidad de ser hombre teniendo gripe

Código Python:

```
pab = [p / pb for p in productos]
```

FUNCIONES

```
# Teorema de Bayes
def bayes(e, b):
    n = float(sum(e))
    pa = [h / n for h in e]
    pi = [k / n for k in b]
    pba = [pi[i] / pa[i] for i in range(len(pi))]
    prods = [pa[i] * pba[i] for i in range(len(pa))]
    ptb = sum(prods)
    pab = [p / pb for p in prods]
    return pab
```

BIBLIOGRAFÍA COMPLEMENTARIA

[0] Probabilidad y Estadística, Murray Spiegel. McGraw-Hill, México 1988. ISBN: 968-451-102-7

ANEXO I: CÁLCULOS COMPLEJOS

ESTADÍSTICA POBLACIONAL Y MUESTRAL: CÁLCULO DE VARIANZA Y DESVÍO ESTÁNDAR

Media

$$\bar{x} = \frac{\sum x_i}{n}$$

Varianza poblacional

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n}$$

Varianza muestral

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

Desvío estándar muestral

$$\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

Desvío estándar poblacional

$$\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}$$

```
from math import sqrt

muestras = [12, 23, 24, 22, 10, 17]  # lista de ejemplo

n = len(muestras)
media = sum(muestras) / float(n)
diferencias = [xi - media for xi in muestras]
potencias = [x ** 2 for x in diferencias]
sumatoria = sum(potencias)

varianza_muestral = sumatoria / (n - 1)
varianza_poblacional = sumatoria / n

desvio_muestral = sqrt(varianza_muestral)
desvio_poblacional = sqrt(varianza_poblacional)
```

PRODUCTO ESCALAR DE DOS VECTORES

```
vector1 = [3, 0]
vvector2 = [4, 3]
pe = sum([x * y for x, y in zip(vector1, vector2)])
```

CÁLCULOS DE FRECUENCIA RELATIVA, ABSOLUTA Y ACUMULADA

```
# FRECUENCIA ABSOLUTA
# Cantidad de veces que un valor aparece en una muestra

muestras = [1, 2, 3, 4, 3, 2, 6, 7, 3, 3, 1, 8, 5, 9]
absolutos = []
frecuencias = []

for n in muestras:
    if not n in absolutos:
        absolutos.append(n)
        fi = muestras.count(n)
        frecuencias.append(fi)

N = sum(frecuencias)  # == len(muestras)


# FRECUENCIA RELATIVA
# Cociente entre la frecuencia absoluta y N
relativas = [float(fi) / N for fi in frecuencias]
sumarelativas = round(sum(relativas))  # == 1


# FRECUENCIA ACUMULADA
# Suma de todas las frecuencias menores o iguales a la frecuencia absoluta
frecuencias.sort()
acumuladas = [sum(frecuencias[:i+1]) for i, fi in enumerate(frecuencias)]


# FRECUENCIA RELATIVA ACUMULADA
# Cociente entre frecuencia acumulada y cantidad total de datos
relacumuladas = [float(f) / N for f in acumuladas]
```

ANEXO II: CREACIÓN DE UN MENÚ DE OPCIONES

En el *scripting*, puede resultar útil, dar al usuario un menú de opciones y hacer que el script, actúe según la opción elegida por el usuario. A continuación, se muestra un truco para resolver esto de forma simple e ingeniosa.

1) Primero es necesario que todo el *script* esté organizado en funciones.

2) En segundo lugar, es necesario que todas las funciones tengan su documentación correspondiente, definiendo qué es exactamente lo que hace la función:

```
def leer_archivo():
    """Leer archivo CSV"""
    return "leer"


def escribir_archivo():
    """Escribir archivo CSV"""
    return "escribir"


def _sumar_numeros(lista):
    """Sumar los números de una lista"""
    return "privada"
```

3) A continuación, se define una lista con el nombre de todas las funciones que serán accesibles por el usuario, desde el menú:

```
funciones = ['leer_archivo', 'escribir_archivo']
```

El truco consistirá en automatizar tanto la generación del menú, como la llamada a la función.

Para **automatizar la generación del menú**, el truco consiste en valerse de:

- La lista del paso 3

- La función locals()
- El atributo __doc__

```
numero = 1  # se usará luego para acceder a la función
menu = "Elija una opción:\n"

for funcion in funciones:
    menu += "\t{}. {}\n".format(numero, locals()[funcion].__doc__)
    numero = numero + 1  # incrementa el número en cada iteración


echo(menu)
opcion = int(get("Su opción: "))
# echo y get: hacks aprendidos en el curso de introducción
```

Finalmente, para **acceder dinámicamente a la función** elegida por el usuario, el truco consistirá en emplear la opción elegida por el usuario, como índice para acceder al nombre de la función desde la lista, y recurrir nuevamente a *locals* para invocar a la función:

```
funcion = funciones[opcion - 1]  # se obtiene el nombre de la función
locals()[funcion]()              # se invoca a la función mediante locals()
```

CERTIFÍCATE

Demuestra cuánto has aprendido!

Si llegaste al final del curso puedes obtener una triple certificación:

- Certificado de **asistencia** (emitido por la escuela **Eugenia Bahit**)
- Certificado de **aprovechamiento** (emitido por **CLA Linux**)
- Certificación **aprobación** (emitido por **LAECI**)

Informáte con tu docente o visita la Web de certificaciones en http://python.eugeniabahit.org.

Si necesitas preparar tu examen, puedes inscribirte en el
Curso de Ciencia de Datos con Python en la
Escuela de Informática Eugenia Bahit
www.eugeniabahit.com