



1

# MISCELÁNEO

## ITERADORES

# MISCELÁNEO

## ■ El protocolo **iterador**

- Es el protocolo que debe cumplir un objeto para ajustarse a las reglas de un **for-in**. Un objeto conforme a este protocolo es un **iterador**.
- Un **iterador** debe proporcionar los métodos:
  - `__iter__()` → Devuelve el objeto. Se invoca una vez.
  - `__next__()` → Devuelve el siguiente valor de la serie o la excepción **StopIteration** si no hay más elementos. Se invoca tantas veces como elementos tenga la serie.

# MISCELÁNEO

## ■ El protocolo **iterador**. Ejemplo:

```
class Iterador:  
    def __init__(self, max) -> None:  
        self.contador = 0  
        self.max = max  
  
    def __iter__(self):  
        return self  
  
    def __next__(self):  
        self.contador+=2  
        if self.contador > self.max:  
            raise StopIteration  
        return int(self.contador)  
  
for i in Iterador(10):  
    print(i)
```

# MISCELÁNEO

## ■ El protocolo **iterador**. Ejercicio:

- Construir una clase Factura con los atributos nombre, importe y estado (pendiente o pagada).
- Construir una clase Administracion que contenga una lista de facturas. De implementar métodos para agregar facturas (siempre en estado pendiente) y pagarlas.
- Construir sobre la clase Administración un iterador que proporcione las facturas pendientes de pago.
- Alternativa: construir dos clases que, a partir de un objeto de Administración, proporcionen iteradores de facturas pendientes y facturas pagadas.

**MISCELÁNEO**

**GENERADORES**

# MISCELÁNEO

## ■ Generador

- Fragmento de código (función) capaz de generar valores.
- También conocidos como **iteradores**.
- La función **range** es un generador.
- Los generadores sustituyen la palabra **return** por la palabra **yield**.
- Se pueden usar en comprensión de listas.

# MISCELÁNEO

## ■ Generador

```
def generar_pares(max):
    for i in range(0,max,2):
        yield i

#Consumo desde bucle for
for i in generar_pares(10):
    print(i)

#Consumo desde comprensión de listas
pares = [par for par in generar_pares(10)]
print(pares)
```

# MISCELÁNEO

- Generadores a partir de comprensión de listas:
  - Repaso de comprensión de listas:

```
lista_numeros = [n for n in range(10)]
lista_pares = [n for n in range(10) if n%2==0]
lista_pares_mod = [n if n%2==0 else 'impar' for n in range(10)]
```

# MISCELÁNEO

- Generadores a partir de comprensión de listas:  
sustituyendo los corchetes por paréntesis se obtienen  
generadores.

```
lista_numeros = [n for n in range(10)]
lista_pares = [n for n in range(10) if n%2==0]
lista_pares_mod = [n if n%2==0 else 'impar' for n in range(10)]

generador_numeros = (n for n in range(10))
generador_pares = (n for n in range(10) if n%2==0)
generador_pares_mod = (n if n%2==0 else 'impar' for n in range(10))
```

# MISCELÁNEO

## FUNCIONES LAMBDA

# MISCELÁNEO

- Función **lambda** o función anónima (función sin nombre).
- Pretende simplificar el código.
- Tiene un único uso.
- Sintaxis: **lambda parámetros : expresión**
  - **expresión** es el retorno de la ejecución de la función

```
two = lambda: 2
sqr = lambda x: x * x
pwr = lambda x, y: x ** y

for a in range(-2, 3):
    print(sqr(a), end=" ")
    print(pwr(a, two()))
```

Fuente del ejemplo: [edube.org](https://edube.org)

# MISCELÁNEO

- Funciones **lambda** y la función **map**.

```
nombres = ["anna", "pep", "iñaki", "nicolás", "lucía"]

#Primeras letras convertidas a mayúsculas
for nombre in map(lambda nombre : nombre.capitalize(), nombres):
    print(nombre)
```

# MISCELÁNEO

- Funciones **lambda** y la función **filter**.

```
nombres = ["anna","pep","iñaki","nicolás","lucía"]

#Nombres que contienen la letra 'a'
for nombre in filter(lambda nombre : 'a' in nombre, nombres):
    print(nombre)
```

# MISCELÁNEO

CLOSURES - CIERRES

# MISCELÁNEO

- Un **cierre (closure)** permite ‘demorar’ la entrega de datos pese a que el contexto en el que se crearon ya no existe.
- Un **cierre** se define como una función (inner) dentro de otra función (outer).
  - La función externa devuelve la función interna, manteniendo esta su contexto.
  - La función interna puede ser invocada en cualquier momento.
  - La función interna no se ejecuta hasta que es invocada.
  - La función externa admite parámetros.
  - La función interna admite parámetros.

# MISCELÁNEO

- Ejemplo de **cierre**:

```
def externa(nombre):
    nombre_mayusculas = nombre.upper()

    def interna():
        return nombre_mayusculas

    return interna

funcion = externa("Estefanía")
print(funcion())
```

# MISCELÁNEO

PROCESANDO ARCHIVOS

# MISCELÁNEO

- **Nombre de archivo canónico:** referencia a un archivo de manera única, lo que permite comparar con éxito referencias al mismo archivo.
- **Referencias a archivos de los distintos SO:**
  - **Windows:**
    - Separador de rutas: \
    - Las unidades tienen identificador: c:\directorio\nombre\_archivo
    - No diferencia mayúsculas de minúsculas
  - **Unix\Linux:**
    - Separador de rutas: /
    - Todo el sistema de archivos cuelta del nodo raíz:  
/directorio/nombre\_archivo:
    - Diferencia mayúsculas de minúsculas

# MISCELÁNEO

- En Windows, los nombres de archivos:

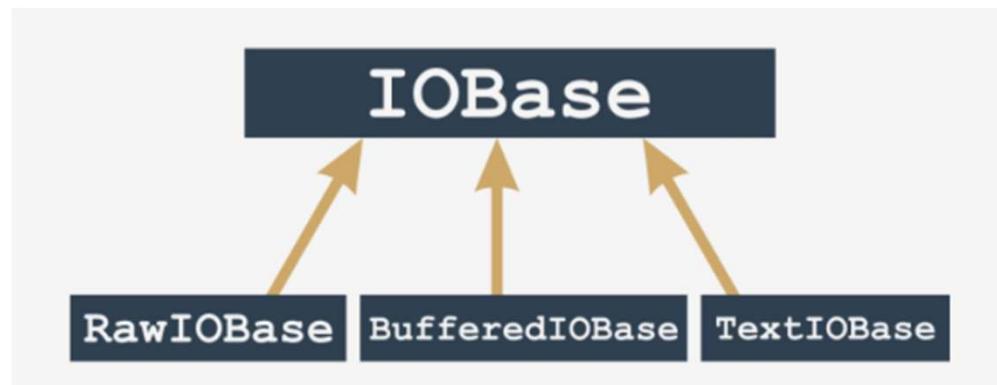
- “c:\carpeta\archivo” → Error
- “c:\\carpeta\\archivo” → OK
- “c:/carpeta/archivo” → OK

# MISCELÁNEO

- La comunicación en Python con los archivos se realiza a través de **handles** (manejadores) o **streams** (canal o flujo).
- Función **open** → apertura.
- Método **close** → cierre.
- Modos de apertura **streams**:
  - Lectura. Si se intenta escribir: **UnsupportedOperation** del módulo **io** y hereda de **OSError** y de **ValueError**.
  - Escritura. Si se intenta leer: **UnsupportedOperation**.
  - Actualizar. Permite leer y escribir.

# MISCELÁNEO

- El modo de apertura (a través de la función **open**) determina el tipo de archivo que se maneja.



Fuente de la figura: [edube.org](https://edube.org)

# MISCELÁNEO

- En función del contenido, los archivos pueden ser:
  - De tipo **texto**.
  - De tipo **binario**.
- En archivos de texto, para garantizar la portabilidad, Python convierte los caracteres **\r\n** de fin de línea de Windows en **\n**, que es el carácter de fin de línea de Unix.

# MISCELÁNEO

## ■ Apertura de ficheros:

- `open(file, mode='r', encoding=None)`
- Mode:
  - ‘r’ → Lectura, debe existir.
  - ‘w’ → Modo escritura, se crea o se trunca si existe. Si no se puede crear, da error.
  - ‘a’ → Actualización (append), se crea o se posiciona el ‘cabezal de escritura’ al final.
  - ‘r+’ → Lectura y actualización, debe existir y poderse escribir. Permite lectura y escritura.
  - ‘w+’ → Escritura y actualización, no necesita existir, el contenido se respeta si existe. Permite operaciones de lectura y escritura.

# MISCELÁNEO

- **Apertura de ficheros (modo texto y binario):**

| Modo texto | Modo binario | Descripción               |
|------------|--------------|---------------------------|
| rt         | rb           | lectura                   |
| wt         | wb           | escritura                 |
| at         | ab           | adjuntar                  |
| r+t        | r+b          | lectura y actualización   |
| w+t        | w+b          | escritura y actualización |

Fuente de la figura: [edube.org](https://edube.org)

# MISCELÁNEO

- Cierre de ficheros:
  - Método **close**.
  - Devuelve **IOError** si hay un error.

# MISCELÁNEO

- Streams preabiertos (no requieren uso de la función **open** ni del método **close**).
  - Forman parte del módulo **sys**.
    - **sys.stdin** → Entrada estándar (stream utilizado por **input**).
    - **sys.stdout** → Salida estándar (stream utilizado por **print**).
    - **sys.stderr** → Salida estándar de error.

# MISCELÁNEO

- Gestión de errores.
- El objeto **IOError**:
  - Tiene la propiedad **errno**, que contiene el número de error, asociado a constantes del módulo **errno**.
  - **errno.EFBIG** → Archivo demasiado grande
  - **errno.EISDIR** → Es un directorio
  - **errno.EMFILE** → Demasiados archivos abiertos
  - **errno.ENOENT** → El archivo o directorio no existe
  - **errno.ENOSPC** → No queda espacio en el dispositivo

# MISCELÁNEO

- Gestión de errores.
- Función **strerror()** del módulo **os**.
- Dado un número de error, proporciona información:

```
import errno
import os
#errno.EISDIR es 21
print(os.strerror(errno.EISDIR))
#Salida 'Is a directory'
```

# MISCELÁNEO

## ■ Lectura:

- Método **read()** → Lee el fichero completo.
- Método **read(número\_caracteres)** → Lee el número de caracteres indicado. Si lee “ ” es que ha llegado al final del fichero.
- Método **readline()** → Lee una línea y avanza. Si lee “ ” es que ha llegado al final del fichero.
- Método **readlines()** → Lee todo el fichero devolviendo una lista con cada línea del mismo.
- Método **readlines(número\_bytes)** → Lee el número de bytes indicado devolviendo una lista con cada línea leída. Si devuelve una lista vacía es que ha llegado al final del fichero.

# MISCELÁNEO

## ■ Lectura:

- La función **open** devuelve un **iterable** que recorre el fichero línea a línea → Se puede recorrer con un bucle **for-in**.
  - El fichero se cierra automáticamente.

```
for line in open("fichero_entrada.txt"):  
    print(line)
```

# MISCELÁNEO

- **Escritura:**

- **El método `write`.**

- Sintaxis: *fichero.write(cadena)*.
    - No agrega salto de línea.

# MISCELÁNEO

## ■ Ficheros binarios:

- La función **bytearray** → permite representar un array de bytes (un objeto **bytearray**).
  - *array\_de\_bytes = bytearray(10)* → Construye un **bytearray** de 10 posiciones lleno con ceros.
  - *array\_de\_bytes = bytearray([8,4,3])* → Construye un **bytearray** de 3 posiciones con los datos de la lista.
  - Admite únicamente bytes (números enteros entre 0 y 255).
    - Si no son enteros: **TypeError**.
    - Si no está dentro del rango: **ValueError**.

# MISCELÁNEO

## ■ Ficheros binarios:

- Los objetos **bytearray**:
  - Son mutables.
  - Se pueden utilizar con la función **len**.
  - Se accede a sus elementos por índice.
  - Para escribir su contenido, abrir fichero utilizando el parámetro **b**.
  - Lectura: utilizando el método **readinto** → Carga la lectura en un **bytearray**.

# MISCELÁNEO

## ■ Ficheros binarios:

### ▪ Ejemplo de lectura y análisis de fichero binario:

```
fichero = open("imagen.bmp", "rb")

signature = bytearray(2)
header = bytearray(12) #La cabecera también debería incluir signature
header_size = bytearray(4)
width = bytearray(4)
height = bytearray(4)

fichero.readinto(signature)
fichero.readinto(header)
fichero.readinto(header_size)
fichero.readinto(width)
fichero.readinto(height)

print(int.from_bytes(width, "little"))#Little Endian

fichero.close()
```

# MISCELÁNEO

## ■ Ficheros binarios:

- Ejemplo de lectura y análisis de fichero binario.
- Formato bmp (fuente <http://www.ece.ualberta.ca/>)

| Name              | Size     | Offset | Descr   |
|-------------------|----------|--------|---|
| <b>Header</b>     | 14 bytes |        | Windows Structure: BITMAPFILEHEADER                               |
| Signature         | 2 bytes  | 0000h  | 'BM'  |
| FileSize          | 4 bytes  | 0002h  | File size in bytes  |
| reserved          | 4 bytes  | 0006h  | unused (=0)   |
| DataOffset        | 4 bytes  | 000Ah  | Offset from beginning of file to the beginning of the bitmap data |
| <b>InfoHeader</b> | 40 bytes |        | Windows Structure: BITMAPINFOHEADER                               |
| Size              | 4 bytes  | 000Eh  | Size of InfoHeader =40  |
| Width             | 4 bytes  | 0012h  | Horizontal width of bitmap in pixels                              |
| Height            | 4 bytes  | 0016h  | Vertical height of bitmap in pixels                               |

# MISCELÁNEO

## ■ Ficheros binarios:

- El método **read** aplicado a ficheros binarios devuelve todo el contenido en un objeto de la clase **bytes**.
- **bytes** es inmutable.
- Se puede convertir a **bytearray**: `bytearray(objeto_bytes)`.
- El método **read** admite como argumento el número de bytes a leer.

# MISCELÁNEO

## ■ Ficheros binarios:

- Ejemplo de uso de método **read** con argumentos:

```
fichero = open("imagen.bmp", "rb")
signature = fichero.read(2)
header = fichero.read(12)
header_size = fichero.read(4)
width = fichero.read(4)
height = fichero.read(4)
print(int.from_bytes(width, "little"))#Little Endian
fichero.close()
```

# MISCELÁNEO

## INTRODUCCIÓN AL MÓDULO os

## MISCELÁNEO

- El módulo **os** permite interactuar con el S.O.:
  - Operaciones con archivos y directorios.
  - Obtener información del S.O.
  - Gestionar procesos.
  - Operar con *streams* de entrada-salida.

# MISCELÁNEO

- Método **os.uname()**. Proporciona un objeto con los siguientes atributos:
  - **systemname**: nombre del sistema operativo.
  - **nodename**: nombre del ordenador en la red.
  - **release**: reléase (versión ‘mayor’) del sistema operativo.
  - **version**: versión (‘menor’) del sistema operativo.
  - **machine**: identificador del hardware.

Nota: En Windows, **uname()** se encuentra en el módulo **platform**.

# MISCELÁNEO

## ■ Manejo de ficheros y directorios:

- Método **os.getcwd()**. Proporciona el directorio actual.
- Método **os.listdir(path)**. Obtiene el contenido de la ruta.
- Método **os.chdir(path)**. Cambia el directorio actual.
- Método **os.mkdir(path)**. Crea un directorio.
  - Si existe: **FileExistsError**.
  - Dispone de un parámetro **mode** para asignar permisos (depende del S.O.).
- Método **os.makedirs(path)**. Crea todos los directorios contenidos en el path. Ejemplo de path: “./d1/d2/d3”.

# MISCELÁNEO

## ■ Manejo de ficheros y directorios:

- Método **os.rename(path, newpath)**. Modifica el nombre de un archivo o directorio. Si existe el nuevo nombre, da un error.
- Método **os.rmdir(path)**. Elimina un directorio.
  - Si no está vacío: **OSError**
  - Si no existe: **FileNotFoundException**
- Método **os.removedirs(path)**. Elimina los directorios de la ruta. Error si no están vacíos o no existen.
- Método **os.remove(filename)**. Elimina un fichero.

# MISCELÁNEO

- Manejo de ficheros y directorios:
  - Método **os.replace(path\_origen, path\_destino)**. Modifica el nombre de un archivo o directorio. Si existe, lo reemplaza.
  - Método **os.stat(path)**. Muestra información sobre un fichero o directorio. Por ejemplo, el tamaño en bytes o la última fecha de modificación.

# MISCELÁNEO

## ■ Manejo de ficheros y directorios:

- Método **os.access(fichero, mode)**. Indica si se dispone de un determinado permiso sobre un fichero.
- Para usar esta función, debes pasarle una de las constantes definidas en el módulo os:
  - **os.F\_OK**: Comprueba si el archivo existe (por defecto).
  - **os.R\_OK**: Comprueba si tienes permiso de lectura.
  - **os.W\_OK**: Comprueba si tienes permiso de escritura (Write).
  - **os.X\_OK**: Comprueba si tienes permiso de ejecución (Execute). En directorios, esto indica si puedes entrar en ellos.

# MISCELÁNEO

- Manejo de procesos.
  - Método **os.system(str)**. Ejecuta directamente sobre el sistema el comando indicado en *str*.
  - Retorna el valor proporcionado por el comando o proceso.

# MISCELÁNEO

## ■ Manejo del entorno.

- Atributo **os.environ** – Diccionario de variables de entorno.
- Método **os.getenv(key, default=None)** – Obtiene una variable de entorno.
- Método **os.putenv(key, value)** – Define una variable de entorno.
- Método **os.getpid()** – Devuelve el ID del proceso actual.
- Método **os.system(command)** – Ejecuta un comando del sistema (uso básico).

# MISCELÁNEO

- **os.walk()** → Recorre recursivamente un árbol de directorios.

```
for root, dirs, files in os.walk("ruta"):  
    print(root)  
    print(dirs)  
    print(files)
```

- **os.fwalk()** → Versión avanzada de os.walk(). En lugar de trabajar con cadenas (str) lo hace con descriptores de fichero.

# MISCELÁNEO

## ■ **módulo os.path**

- **os.path.exists(path)** – Verifica si una ruta existe.
- **os.path.isfile(path)** – Verifica si es un archivo.
- **os.path.isdir(path)** – Verifica si es un directorio.
- **os.path.join(\*paths)** – Une rutas de forma portable (para compatibilidad de rutas Linux/Windows).
- **os.path.abspath(path)** – Obtiene la ruta absoluta.
- **os.path.basename(path)** – Devuelve el nombre del archivo.
- **os.path.dirname(path)** – Devuelve el directorio padre.
- **os.path.splitext(path)** – Separa ruta y nombre de fichero.

# MISCELÁNEO

## INTRODUCCIÓN AL MÓDULO `datetime`

# MISCELÁNEO

- Clase **datetime.date**.

- Contiene una fecha.

- Atributos:

- **year** (entre MINYEAR -1- y MAXYEAR -9999-)
    - **month** (entre 1 y 12)
    - **day** (entre 1 y el último día del mes y año indicados)

- Constructor:

- **date(año,mes,día)**

- Método **today()**. Obtiene la fecha del sistema.

# MISCELÁNEO

- Clase **datetime.date**.

- Construcción a partir de una marca de tiempo (*timestamp*).

- Tiempo transcurridos desde el 1/1/1970.

- Se obtiene mediante el método **time()** del módulo **time**:

```
import time  
timestamp = time.time()
```

- El método **fromtimestamp** genera un objeto **date** a partir del *timestamp*.

# MISCELÁNEO

- Clase **datetime.date**.
  - Construcción a partir del formato **ISO**.
    - Método **fromisoformat(YYYY-MM-DD)** → Compatible con el estándar ISO 8601.
      - Completar mes y día con cero si sólo tiene un dígito.
      - `d = date.fromisoformat('2023-06-21')`
    - Método **replace(year, month, day)** → Reemplaza uno o más valores de una fecha y **devuelve** el objeto date modificado.

# MISCELÁNEO

- Clase **datetime.date**.
  - Método **weekday()** → Devuelve el ordinal del día de la semana, siendo 0 el lunes.
  - Método **isoweekday()** → Devuelve el ordinal del día de la semana, siendo 1 el lunes.

# MISCELÁNEO

- Clase **datetime.time**.

- Constructor:

- `time(hour, minute, second, microsecond, tzinfo, fold)`
    - `tzinfo` → zona horaria
    - `fold` → relacionado con las duplicidades que se producen con los cambios horarios.

# MISCELÁNEO

- Clase **datetime.datetime**.

- Constructor:

- `datetime(year, month, day, hour, minute, second, microsecond, tzinfo, fold)`
      - `tzinfo` → zona horaria
      - `fold` → relacionado con las duplicidades que se producen con los cambios horarios.

# MISCELÁNEO

## ■ Clase **datetime.datetime**.

### ■ Métodos:

- **datetime.today()** → Proporciona un objeto **datetime** con la fecha y hora actual.
- **datetime.now()** → Proporciona un objeto **datetime** con la fecha y hora actual. Admite time-zone.
- **datetime.utcnow()** → Proporciona un objeto **datetime** con la fecha y hora UTC.
- **datetime.date()** → Proporciona la fecha (objeto date) de un objeto **datetime** construido.
- **datetime.time()** → Proporciona la hora (objeto time) de un objeto **datetime** construido.

# MISCELÁNEO

## ■ Clase **datetime.datetime**.

### ■ Métodos:

- **datetime.timestamp()** → Proporciona el *timestamp* de un objeto **datetime** construido.
- **datetime.strftime(formato)** → Permite dar formato a objetos **datetime**, **time** y **date** de cara a su presentación.
- *formato* es una cadena compuesta por directivas y símbolos constantes:
  - Directivas:  
<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>

# MISCELÁNEO

## ■ Clase **datetime.datetime**.

### ■ Métodos:

#### ■ **datetime.strftime(formato)** → Ejemplos

```
d = date(2023, 1, 14)  
print(d.strftime('%d/%m/%Y'))
```

```
t = time(20, 51)  
print(t.strftime("%H:%M:%S"))
```

```
dt = datetime(2023, 3, 22, 20, 51)  
print(dt.strftime("%d/%m/%Y %H:%M:%S"))
```

# MISCELÁNEO

- Clase **datetime.datetime**.

- Métodos:

- **datetime.strptime(str, formato)** → obtiene un objeto **datetime** a partir de una cadena.

- Ejemplo:

```
print(datetime.strptime("15/10/2023 09:10:15", "%d/%m/%Y  
%H:%M:%S"))
```

# MISCELÁNEO

## ■ Clase **datetime.datetime**.

- Operaciones:

- El operador de resta (-) se puede aplicar a objetos **datetime** y **date**.

- Proporciona un objeto de la clase **datetime.timedelta**.

- **No se puede aplicar sobre objetos time.**

- Clase **datetime.timedelta**. Representa un periodo de tiempo.

- Constructor:

- `datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)`

- Atributos:

- `days, seconds, microseconds`.

# MISCELÁNEO

- Clase **datetime.datetime**.
  - Clase **datetime.timedelta**.
  - Operaciones:
    - Suma de timedelta a datetime.
    - Multiplicación de timedelta \* entero.

# MISCELÁNEO

## INTRODUCCIÓN AL MÓDULO time

# MISCELÁNEO

- Contenido de **time.struct\_time** (se puede acceder por índice, excepto a **tm\_zone** y **tm\_gmoff**)

**tm\_year** # Especifica el año.

**tm\_mon** # Especifica el mes (valor de 1 a 12)

**tm\_mday** # Especifica el día del mes (valor de 1 a 31)

**tm\_hour** # Especifica la hora (valor de 0 a 23)

**tm\_min** # Especifica el minuto (valor de 0 a 59)

**tm\_sec** # Especifica el segundo (valor de 0 a 61)

**tm\_wday** # Especifica el día de la semana (valor de 0 a 6)

**tm\_yday** # Especifica el día del año (valor de 1 a 366)

**tm\_isdst** # Especifica si se aplica el horario de verano (1: sí, 0: no, -1: no se sabe)

**tm\_zone** # Especifica el nombre de la zona horaria (valor en forma abreviada)

**tm\_gmtoff** # Especifica el desplazamiento al este del UTC (valor en segundos)

# MISCELÁNEO

- Función **time.asctime(timestamp)**. Convierte un *time* en una cadena de texto:

```
ahora = time.time()  
ahora_gmt = time.gmtime(ahora)  
print(time.asctime(ahora_gmt))
```

- Función **time.mktime(info)**. Convierte *info* (struct-time o tupla) en segundos desde la ‘época de Unix’

# MISCELÁNEO

- Método **time.sleep(segundos)**. Bloquea el hilo de ejecución el número de segundos (entero o float) indicado.
- Función **strftime**. Permite proporcionar una salida con formato de un objeto time.
  - Directivas:
    - <https://docs.python.org/3/library/time.html#time.strftime>
- Función **strptime**. Convierte una cadena a objeto **struct\_time**

# MISCELÁNEO

## INTRODUCCIÓN AL MÓDULO calendar

# MISCELÁNEO

## ■ Constantes:

| Día de la semana | Valor entero | Constante          |
|------------------|--------------|--------------------|
| Lunes            | 0            | calendar.MONDAY    |
| Martes           | 1            | calendar.TUESDAY   |
| Miércoles        | 2            | calendar.WEDNESDAY |
| Jueves           | 3            | calendar.THURSDAY  |
| Viernes          | 4            | calendar.FRIDAY    |
| Sábado           | 5            | calendar.SATURDAY  |
| Domingo          | 6            | calendar.SUNDAY    |

Fuente de la tabla: [edube.org](http://edube.org)

# MISCELÁNEO

- Función **calendar.calendar(año)** → Proporciona un **str** con el calendario del año indicado.
  - Parámetros:
    - w: ancho de la columna de fecha (por defecto 2)
    - l: número de líneas por semana (por defecto 1)
    - c: número de espacios entre las columnas del mes (por defecto 6)
    - m: número de columnas (por defecto 3)
- Función **calendar.prcal(año)** → Igual que calendar, pero mostrando el resultado por consola.

# MISCELÁNEO

- Función **calendar.month**(año, mes) → Proporciona un **str** con el calendario del año y mes indicado. Dispone de los mismo parámetros que **calendar**.
- Función **calendar.prmonth**(año,mes) → Igual que **month**, mostrando por pantalla.
- Función **calendar.setfirstweekday**(Calendar.día). Asigna el día indicado como primer día de la semana:
  - `calendar.setfirstweekday(calendar.SUNDAY)`
- Función **calendar.weekday**(year,month,day). Obtiene un número entre 0 y 6 correspondiente al día de la semana.
- Función **calendar.weekheader**(numero\_caracteres). Devuelve un string con los encabezados de los días de la semana.

# MISCELÁNEO

- Función **calendar.isleap(año)** → Indica si el año es bisiesto.
- Función **calendar.leapdays(año0, año1)** → Indica el número de años bisestos en el rango de años.

# MISCELÁNEO

- Clases para crear calendarios:
  - `calendar.Calendar`
  - `calendar.TextCalendar`
    - `calendar.LocalTextCalendar`
  - `calendar.HTMLCalendar`
    - `calendar.LocalHTMLCalendar`
- Una vez construido, utilizar los métodos **formatxxx** para construir el calendario.

```
c = calendar.HTMLCalendar()  
texto_html = c.formatyear(2023)  
print(texto_html)
```

# MISCELÁNEO

- Construcción de calendarios:
  - Recibe como parámetro el primer día de la semana (por defecto 0): `calendar.Calendar(firstweekday)`
    - Usar números o las constantes de `calendar`.

# MISCELÁNEO

- Iterar un calendario:

- Se puede iterar por los días del calendario con el método **iterweekdays**, que recorre dichos días (números enteros) comenzando por **firstweekday**:

```
c = calendar.Calendar(calendar.SUNDAY)
for weekday in c.iterweekdays():
    print(weekday, end=" ")
```

- Se puede iterar por los meses del calendario con el método **itermonthdates(año, mes)**. Devuelve objetos de la clase **datetime.date** e incluye los días anteriores y posteriores al mes si son necesarios para completar semanas.
- Método **itermonthdays(año, mes)**. Devuelve los días del mes (int). Rellena con ceros los días de la primera semana y de la última si hace falta para completar semana.

# MISCELÁNEO

- Iterar un calendario:

- Método **itermonthdates2**: devuelve tuplas con el día del mes y el número de día de la semana.
- Método **itermonthdates3**: devuelve tuplas con un año, un mes y un día de los números del mes. Desde la versión 3.7 de Python.
- Método **itermonthdates4**: devuelve tuplas con números de año, mes, día del mes y día de la semana. Desde la versión 3.7 de Python.

- Otros métodos:

- Método **monthdays2calendar**: devuelve una lista de semanas. Cada semana está compuesta por 7 tuplas (una por cada día de la semana) con el número de día de la semana (empezando por 0) y el número del día de mes.