Módulo 4 (Intermedio): Biblioteca datetime

Introducción al módulo datetime

En esta sección, aprenderás sobre un módulo de Python llamado datetime.

Como puedes adivinar, proporciona **clases para trabajar con la fecha y hora**. Si crees que no necesitas profundizar en este tema, hablemos de ejemplos del uso de la fecha y la hora en la programación.

La fecha y la hora tienen innumerables usos y probablemente sea difícil encontrar una aplicación de producción que no los utilice. A continuación, se muestran algunos ejemplos:

- **Registro de eventos**: gracias al conocimiento de la fecha y la hora, podemos determinar cuándo ocurre exactamente un error crítico en nuestra aplicación. Al crear registros, puedes especificar el formato de fecha y hora.
- **Seguimiento de cambios en la base de datos**: a veces es necesario almacenar información sobre cuándo se creó o modificó un registro. El módulo datetime será perfecto para este caso.
- **Validación de datos**: pronto aprenderás a leer la fecha y hora actuales en Python. Conociendo la fecha y hora actuales, podrás validar varios tipos de datos, por ejemplo, si un cupón de descuento ingresado por un usuario en nuestra aplicación sigue siendo válido.
- **Almacenamiento de información importante**: ¿te imaginas las transferencias bancarias sin almacenar la información de cuándo se realizaron? La fecha y la hora de ciertas acciones deben conservarse y debemos ocuparnos de ello.

La fecha y la hora se utilizan en casi todas las áreas de nuestras vidas, por lo que es importante familiarizarse con el módulo datetime de Python. ¿Estás listo para una nueva dosis de conocimiento?

Obtener la fecha local actual y crear objetos del tipo fecha

Una de las clases proporcionadas por el módulo *datetime* es una clase llamada *date*. Los objetos de esta clase representan una fecha que consta de año, mes y día. Mira el código en el editor para ver cómo se ve en la práctica y como obtener la fecha local actual usando el método today.

```
from datetime import date

today = date.today()

print("Hoy:", today)
print("Año:", today.year)
print("Mes:", today.month)
print("Día:", today.day)
```

El método *today* devuelve un objeto del tipo date que representa la fecha local actual. Toma en cuenta que el objeto date tiene tres atributos: año, mes y día.

Ten cuidado, porque estos atributos son de solo lectura. Para crear un objeto date, debes pasar los parámetros año, mes y día de la siguiente manera:

```
from datetime import date

my_date = date(2019, 11, 4)
```

```
print(my_date)
```

Al crear un objeto date, toma en cuenta las siguientes restricciones:

Parámetro	Restricciones
`año`	El parámetro *año* debe ser mayor o igual a 1 (constante MINYEAR) y menor o igual a 9999 (constante MAXYEAR).
`mes`	El parámetro *mes* debe ser mayor o igual a 1 y menor o igual a 12.
`día`	El parámetro *día* debe ser mayor o igual a 1 y menor o igual que el último día del mes y año indicados.

Nota: Más adelante en este curso, aprenderás a cambiar el formato de fecha predeterminado.

Creación de un objeto de fecha a partir de una marca de tiempo

La clase *date* nos da la capacidad de crear un objeto del tipo fecha a partir de una marca de tiempo.

En Unix, la marca de tiempo expresa el número de segundos desde el 1 de Enero de 1970 a las 00:00:00 (UTC). Esta fecha se llama la **época Unix**, porque es cuando comenzó el conteo del tiempo en los sistemas Unix.

La marca de tiempo es en realidad la diferencia entre una fecha en particular (incluida la hora) y el 1 de enero de 1970, 00:00:00 (UTC), expresada en segundos.

Para crear un objeto de fecha a partir de una marca de tiempo, debemos pasar una marca de tiempo Unix al método *fromtimestamp*.

Para este propósito, podemos usar el módulo *time*, que proporciona funciones relacionadas con el tiempo. Uno de ellos es una función llamada `time()`, que devuelve el número de segundos desde el 1 de enero de 1970 hasta el momento actual en forma de número flotante.

```
from datetime import date
import time

timestamp = time.time()
print("Marca de tiempo:", timestamp)

d = date.fromtimestamp(timestamp)
print("Fecha:", d)
```

Si ejecutas el código de muestra varias veces, podrás ver cómo se incrementa la marca de tiempo. Vale la pena agregar que el resultado de la función time depende de la plataforma, porque en los sistemas Unix y Windows, **los segundos intercalares no se cuentan**.

Nota: En esta parte del curso también hablaremos sobre el módulo time.

Creando un objeto de fecha usando el formato ISO

El módulo *datetime* proporciona varios métodos para crear un objeto date. Uno de ellos es el método *fromisoformat*, que toma una fecha en el formato **AAAA-MM-DD** compatible con el estándar ISO 8601.

El estándar ISO 8601 define cómo se representan la fecha y la hora. Se usa a menudo, por lo que vale la pena tomarse un momento para familiarizarse con él. Echa un vistazo a la imagen que describe los valores requeridos por el formato:




```
from datetime import date

d = date.fromisoformat('2019-11-04')
print(d)
```

En nuestro ejemplo, AAAA es 2019, MM es 11 (noviembre) y DD es 04 (cuarto de noviembre).

Cuando sustituyas la fecha, asegúrate de agregar 0 antes de un mes o de un día expresado por un número menor que 10.

Nota: El método *fromisoformat* ha estado disponible en Python desde la versión 3.7.

El método replace()

A veces, es posible que debas reemplazar el año, el mes o el día con un valor diferente. No puedes hacer esto con los atributos de año, mes y día porque son de solo lectura. En este caso, puedes utilizar el método llamado replace.

```
from datetime import date

d = date(1991, 2, 5)
print(d)

d = d.replace(year=1992, month=1, day=16)
print(d)
```

Resultado:

```
1991-02-05
1992-01-16
```

Los parámetros year, month y day son opcionales. Puedes pasar solo un parámetro al método *replace*, por ejemplo, año, o los tres como en el ejemplo.

El método *replace* devuelve un objeto date modificado, por lo que debes recordar asignarlo a alguna variable.

¿Que dia de la semana es?

Uno de los métodos más útiles que facilita el trabajo con fechas es el método llamado *weekday*. Devuelve el día de la semana como un número entero, donde 0 es el Lunes y 6 es el Domingo.

```
from datetime import date

d = date(2019, 11, 4)
print(d.weekday())
```

Resultado:

```
0
```

La clase *date* tiene un método similar llamado *isoweekday*, que también devuelve el día de la semana como un número entero, pero 1 es Lunes y 7 es Domingo:

```
from datetime import date

d = date(2019, 11, 4)
print(d.isoweekday())
```

Resultado:

```
1
```

Como puedes ver, para la misma fecha obtenemos un número entero diferente, pero expresando el mismo día de la semana. El entero devuelto por el método *isodayweek* sigue la especificación ISO 85601.

Creando objetos time

Ya sabes cómo presentar una fecha utilizando el objeto *date*. El módulo *datetime* también tiene una clase que te permite presentar la hora. ¿Puedes adivinar su nombre? Sí, se llama *time*:

```
time(hour, minute, second, microsecond, tzinfo, fold)
```

El constructor de la clase time acepta los siguientes parámetros opcionales:

Parámetro	Restricciones
`hour`	El párametro *hour* debe ser mayor o igual que 0 y menor que 23.
`minute`	El párametro *minute* debe ser mayor o igual que 0 y menor que 59.
`second`	El párametro *second* debe ser mayor o igual que 0 y menor que 59.
`microsecond`	El párametro *microsecond* debe ser mayor o igual que 0 y menor que 1000000.
`tzinfo`	El párametro *tzinfo* debe ser un objeto de la subclase `tzinfo` o `None` (por defecto).
`fold`	El párametro *fold* debe ser 0 o 1 (predeterminadamente 0).

El párametro *tzinfo* está asociado con las zonas horarias, mientras que *fold* está asociado con el tiempo de pared. No los usaremos durante este curso, pero te recomendamos que te familiarices con ellos.

Veamos cómo crear un objeto de tiempo en la práctica.

```
from datetime import time

t = time(14, 53, 20, 1)

print("Tiempo:", t)
print("Hora:", t.hour)
print("Minuto:", t.minute)
print("Segundo:", t.second)
print("Microsegundo:", t.microsecond)
```

Resultado:

```
Tiempo: 14:53:20.000001
Hora: 14
Minuto: 53
Segundo: 20
Microsegundo: 1
```

En el ejemplo, pasamos cuatro parámetros al constructor de la clase: hour, minute, second, and microsecond. Se puede acceder a cada uno de ellos utilizando los atributos de clase.

El módulo time

Además de la clase *time*, la biblioteca estándar de Python ofrece un módulo llamado *time*, que proporciona una función relacionada con el tiempo. Ya se tuvo la oportunidad de aprender la función llamada *time* cuando se habló de la clase *date*. Ahora veremos otra función útil disponible en este módulo.

Debes pasar muchas horas frente a una computadora mientras realiza este curso. A veces puedes sentir la necesidad de tomar una siesta. ¿Por qué no? Escribamos un programa que simule la siesta corta de un estudiante.

```
import time
```

```
class Student:
    def take_nap(self, seconds):
        print("Estoy muy cansado. Tengo que echarme una siesta. Hasta luego.")
        time.sleep(seconds)
        print("¡Dormí bien! ¡Me siento genial!")

student = Student()
student.take_nap(5)
```

Resultado:

```
Estoy muy cansado. Tengo que echarme una siesta. Hasta luego.
¡Dormí bien! ¡Me siento genial!
```

La parte más importante del código de muestra es el uso de la función *sleep* (sí, puedes recordarla de una de las prácticas de laboratorio anteriores en el curso), que suspende la ejecución del programa por el determinado número de segundos.

En nuestro ejemplo, son 5 segundos. Tienes razón, es una siesta muy corta.

Extiende el sueño del estudiante cambiando la cantidad de segundos. Toma en cuenta que la función *sleep* acepta solo un número entero o de punto flotante.

La función ctime()

El módulo *time* proporciona una función llamada *ctime*, que **convierte el tiempo en segundos desde el 1 de enero de 1970 (época Unix) en una cadena**.

¿Recuerdas el resultado de la función *time*? Eso es lo que necesitas pasar a *ctime*.

```
import time

timestamp = 1572879180
print(time.ctime(timestamp))
```

Resultado:

```
Mon Nov  4 14:53:00 2019
```

La función *ctime* devuelve una cadena para la marca de tiempo pasada. En nuestro ejemplo, la marca de tiempo expresa el 4 de noviembre de 2019 a las 14:53:00.

También es posible llamar a la función *ctime* sin especificar el tiempo en segundos. En este caso, se devolverá la hora actual:

```
import time
print(time.ctime())
```

Las funciones gmtime() y localtime()

Algunas de las funciones disponibles en el módulo *time* requieren conocimiento de la clase *struct_time*, pero antes de conocerlas, veamos cómo se ve la clase:

```
time.struct_time:
    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)
```

La clase *struct_time* también permite el acceso a valores usando índices. El índice 0 devuelve el valor en tm_year, mientras que 8 devuelve el valor en tm_isdst.

Las excepciones son *tm_zone* y *tm_gmoff*, a las que no se puede acceder mediante índices. Veamos cómo usar la clase *struct_time* en la práctica.

```
import time

timestamp = 1572879180
print(time.gmtime(timestamp))
print(time.localtime(timestamp))
```

Resultado:

```
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53,
tm_sec=0, tm_wday=0, tm_yday=308, tm_isdst=0)
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53,
tm_sec=0, tm_wday=0, tm_yday=308, tm_isdst=0)
```

El ejemplo muestra dos funciones que convierten el tiempo transcurrido desde la época Unix al objeto struct_time. La diferencia entre ellos es que la función *gmtime* devuelve el objeto struct_time en UTC, mientras que la función *localtime* devuelve la hora local. Para la función *gmtime*, el atributo tm_isdst es siempre 0.

Las funciones asctime() y mktime()

El módulo *time* tiene funciones que esperan un objeto struct_time o una tupla que almacena valores de acuerdo con los índices presentados cuando se habla de la clase *struct_time*.

```
import time

timestamp = 1572879180
st = time.gmtime(timestamp)

print(time.asctime(st))
```

```
print(time.mktime((2019, 11, 4, 14, 53, 0, 0, 308, 0)))
```

Resultado:

```
Mon Nov  4 14:53:00 2019
1572879180.0
```

La primera de las funciones, llamada *asctime*, convierte un objeto *struct_time* o una tupla en una cadena. Toma en cuenta que la conocida función *gmtime* se usa para obtener el objeto struct_time. Si no se proporciona un argumento a la función *asctime*, se utilizará el tiempo devuelto por la función localtime.

La segunda función llamada *mktime* convierte un objeto *struct_time* o una tupla que expresa la hora local al número de segundos desde la época de Unix. En nuestro ejemplo, le pasamos una tupla, que consta de los siguientes valores:

```
2019 => tm_year
11 => tm_mon
4 => tm_mday
14 => tm_hour
53 => tm_min
0 => tm_sec
0 => tm_wday
308 => tm_yday
0 => tm_isdst
```

Creación de objetos datetime

En el módulo *datetime*, la fecha y la hora se pueden representar como objetos separados o como un solo objeto. La clase que combina fecha y hora se llama *datetime*.

```
datetime(year, month, day, hour, minute, second, microsecond, tzinfo, fold)
```

Su constructor acepta los siguientes parámetros:

Parámetro	Restricciones
`year`	El parámetro *year* debe ser mayor o igual a 1 (constante MINYEAR) y menor o igual a 9999 (constante MAXYEAR).
`month`	El parámetro *month* debe ser mayor o igual a 1 y menor o igual a 12.
`day`	El parámetro *day* debe ser mayor o igual a 1 y menor o igual al último día del mes y año indicados.
`hour`	El parámetro *hour* debe ser mayor o igual que 0 y menor que 23.
`minute`	El parámetro *minute* debe ser mayor o igual que 0 y menor que 59.
`second`	El parámetro *second* debe ser mayor o igual que 0 y menor que 59.
`microsecond`	El parámetro *microsecond* debe ser mayor o igual que 0 y menor que 1000000.
`tzinfo`	El parámetro *tzinfo* debe ser una subclase del objeto `tzinfo` o `None` (de manera predeterminada).
`fold`	El parámetro *fold* debe ser 0 o 1 (predeterminadamente 0).

Ahora echemos un vistazo al código en el editor para ver cómo creamos un objeto del tipo datetime.

Resultado:

```
Fecha y Hora: 2019-11-04 14:53:00
Fecha: 2019-11-04
Hora: 14:53:00
```

El ejemplo crea un objeto *datetime* que representa el 4 de noviembre de 2019 a las 14:53:00. Todos los parámetros pasados al constructor van a atributos de clase de solo lectura. Son year, month, day, hour, minute, second, microsecond, tzinfo, y fold.

El ejemplo muestra dos métodos que devuelven dos objetos diferentes. El método llamado *date* devuelve el objeto date con el año, mes y día dados, mientras que el método llamado *time* devuelve el objeto time con la hora y minuto dados.

Métodos que devuelven la fecha y hora actuales

La clase datetime tiene varios métodos que devuelven la fecha y hora actuales. Estos métodos son:

- today(): devuelve la fecha y hora local actual con el atributo tzinfo establecido a None.
- now(): devuelve la fecha y hora local actual igual que el método today, a menos que le pasemos el argumento opcional tz. El argumento de este método debe ser un objeto de la subclase tzinfo.
- utcnow(): devuelve la fecha y hora UTC actual con el atributo tzinfo establecido a None.

```
from datetime import datetime

print("hoy:", datetime.today())
```

```
print("ahora:", datetime.now())
print("utc_ahora:", datetime.utcnow())
```

Como puedes ver, el resultado de los tres métodos es el mismo. Las pequeñas diferencias se deben al tiempo transcurrido entre llamadas posteriores.

Nota: Puedes leer más sobre los objetos *tzinfo* en la documentación.

El obtener una marca de tiempo

Existen muchos convertidores disponibles en Internet que pueden calcular una marca de tiempo en función de una fecha y hora determinadas, pero ¿cómo podemos hacerlo en el módulo *datetime*?

Esto es posible gracias al método *timestamp* proporcionado por la clase datetime.

```
from datetime import datetime

dt = datetime(2020, 10, 4, 14, 55)
print("Marca de tiempo:", dt.timestamp())
```

Resultado:

```
Timestamp: 1601823300.0
```

El método *timestamp* devuelve un valor flotante que expresa el número de segundos transcurridos entre la fecha y la hora indicadas por el objeto datetime y el 1 de enero de 1970, 00:00:00 (UTC).

Formato de fecha y hora

Todas las clases del módulo *datetime* presentadas hasta ahora tienen un método llamado *strftime*. Este es un método muy importante, porque nos permite devolver la fecha y la hora en el formato que especificamos.

El método *strftime* toma solo un argumento en forma de cadena que especifica un formato que puede constar de directivas.

Una directiva es una cadena que consta del carácter % (porcentaje) y una letra minúscula o mayúscula. Por ejemplo, la directiva %Y significa el año con el siglo como número decimal. Veámoslo en un ejemplo.

```
from datetime import date

d = date(2020, 1, 4)
print(d.strftime('%Y/%m/%d'))
```

Resultado:

```
2020/01/04
```

En el ejemplo, hemos pasado un formato que consta de tres directivas separadas por / (diagonal) al método *strftime*. Por supuesto, el carácter separador se puede reemplazar por otro carácter, o incluso por una cadena.

Puedes poner cualquier carácter en el formato, pero solo las directivas reconocibles se reemplazarán con los

valores apropiados. En nuestro formato, hemos utilizado las siguientes directivas:

- **%Y**: devuelve el año con el siglo como número decimal. En nuestro ejemplo, esto es 2020.
- **%m**: devuelve el mes como un número decimal con relleno de ceros. En nuestro ejemplo, es 01.
- **%d**: devuelve el día como un número decimal con relleno de ceros. En nuestro ejemplo, es 04.

Nota: Puedes encontrar todas las directivas disponibles aquí.

El formato de hora funciona de la misma forma que el formato de fecha, pero requiere el uso de directivas adecuadas.

```
from datetime import time
from datetime import datetime

t = time(14, 53)
print(t.strftime("%H:%M:%S"))

dt = datetime(2020, 11, 4, 14, 53)
print(dt.strftime("%y/%B/%d %H:%M:%S"))
```

Resultado:

```
14:53:00
20/November/04 14:53:00
```

El primero de los formatos utilizados se refiere solo al tiempo. Como puedes adivinar, %H devuelve la hora como un número decimal con relleno de ceros, %M devuelve el minuto como un número decimal con relleno de ceros, mientras que %S devuelve el segundo como un número decimal con relleno de ceros. En nuestro ejemplo, %H se reemplaza por 14, %M por 53 y %S por 00.

El segundo formato utilizado combina directivas de fecha y hora. Hay dos nuevas directivas, %Y y %B. La directiva %Y devuelve el año sin siglo como un número decimal con relleno de ceros (en nuestro ejemplo es 20). La directiva %B devuelve el mes como el nombre completo.

En general, tienes mucha libertad para crear formatos, pero debes recordar usar las directivas correctamente. Como ejercicio, puedes comprobar qué sucede si, por ejemplo, intentas utilizar la directiva %Y en el formato pasado al método strftime del objeto time. Intenta averiguar por qué se obtuvo este resultado. ¡Buena suerte!

La función strftime() en el módulo time

Probablemente no te sorprendas al saber que la función strftime está disponible en el módulo time. Se diferencia ligeramente de los métodos strftime en las clases proporcionadas por el módulo datetime porque, además del argumento de formato, también puede tomar (opcionalmente) un objeto tuple o struct_time.

Si no se pasa una tupla o un objeto struct_time, el formateo se realizará utilizando la hora local actual.

```
import time

timestamp = 1572879180
st = time.gmtime(timestamp)

print(time.strftime("%Y/%m/%d %H:%M:%S", st))
print(time.strftime("%Y/%m/%d %H:%M:%S"))
```

Nuestro resultado es el siguiente:

```
2019/11/04 14:53:00
2020/10/12 12:19:40
```

La creación de un formato tiene el mismo aspecto que para los métodos strftime en el módulo datetime. En nuestro ejemplo, usamos %Y, %m, %d, %H, %M y %S directivas que ya conoces.

En la primera llamada a la función, formateamos el objeto struct_time, mientras que en la segunda llamada (sin el argumento opcional), formateamos la hora local. Puede encontrar todas las directivas disponibles en el módulo time aquí.

El método strptime()

Saber cómo crear un formato puede ser útil cuando se usa un método llamado *strptime* en la clase *datetime*. A diferencia del método *strftime*, crea un objeto datetime a partir de una cadena que representa una fecha y una hora.

El método *strptime* requiere que especifiques el formato en el que guardaste la fecha y la hora.

```
from datetime import datetime
print(datetime.strptime("2019/11/04 14:53:00", "%Y/%m/%d %H:%M:%S"))
```

Resultado:

```
2019-11-04 14:53:00
```

En el ejemplo, hemos especificado dos argumentos obligatorios. El primero es una fecha y hora como una cadena: «2019/11/04 14:53:00», mientras que el segundo es un formato que facilita el análisis a un objeto datetime. Ten cuidado, porque si el formato que se especifica no coincide con la fecha y la hora en la cadena, generará un excepción ValueError.

Nota: En el módulo time, puedes encontrar una función llamada strptime, que analiza una cadena que representa un tiempo en un objeto struct_time. Su uso es análogo al método strptime en la clase datetime:

```
import time
print(time.strptime("2019/11/04 14:53:00", "%Y/%m/%d %H:%M:%S"))
```

Su resultado será el siguiente:

```
time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53,
tm_sec=0, t
```

Operaciones de fecha y hora

Tarde o temprano tendrás que realizar algunos cálculos sobre la fecha y la hora. Afortunadamente, existe una clase llamada timedelta en el módulo datetime que se creó con tal propósito.

Para crear un objeto timedelta, simplemente realiza una resta en los objetos date o datetime, tal como hicimos en el ejemplo en el editor.

```
from datetime import date
```

```
from datetime import datetime

d1 = date(2020, 11, 4)
d2 = date(2019, 11, 4)

print(d1 - d2)

dt1 = datetime(2020, 11, 4, 0, 0, 0)
dt2 = datetime(2019, 11, 4, 14, 53, 0)

print(dt1 - dt2)
```

Resultado:

```
366 days, 0:00:00
365 days, 9:07:00
```

El ejemplo muestra la resta para los objetos *date* y *datetime*. En el primer caso, recibimos la diferencia en días, que es de 366 días. Toma en cuenta que también se muestra la diferencia en horas, minutos y segundos. En el segundo caso, recibimos un resultado diferente, porque especificamos el tiempo que se incluyó en los cálculos. Como resultado, recibimos 365 días, 9 horas y 7 minutos.

En un momento aprenderás más sobre la creación de los objetos *timedelta* y sobre las operaciones que puedes realizar con ellos.

Creación de objetos timedelta

Ya has aprendido que un objeto *timedelta* puede devolverse como resultado de restar dos objetos *date* o *datetime*.

Por supuesto, también puedes crear un objeto tu mismo. Para ello, vamos a familiarizarnos con los argumentos aceptados por el constructor de la clase, que son:days, seconds, microseconds, milliseconds, minutes, hours, y weeks. Cada uno de ellos es opcional y el valor predeterminado es 0.

Los argumentos deben ser números enteros o de punto flotante, y pueden ser positivos o negativos.

```
from datetime import timedelta

delta = timedelta(weeks=2, days=2, hours=3)
print(delta)
```

Resultado:

```
16 days, 3:00:00
```

El resultado de 16 días se obtiene convirtiendo el argumento weeks en días (2 semanas = 14 días) y agregando el argumento days (2 días). Este es un comportamiento normal, porque el objeto timedelta solo almacena días, segundos y microsegundos internamente. De manera similar, el argumento hora se convierte en minutos. Echa un vistazo al siguiente ejemplo:

```
from datetime import timedelta

delta = timedelta(weeks=2, days=2, hours=3)
print("Días:", delta.days)
```

```
print("Segundos:", delta.seconds)
print("Microsegundos:", delta.microseconds)
```

Resultado:

```
Días: 16
Segundos: 10800
Microseconds: 0
```

El resultado de 10800 se obtiene convirtiendo 3 horas en segundos. De esta forma el objeto timedelta almacena los argumentos pasados durante su creación. Las semanas se convierten en días, las horas y los minutos en segundos y los milisegundos en microsegundos.

Ya sabes cómo el objeto *timedelta* almacena los argumentos pasados internamente. Veamos cómo se puede utilizar en la práctica.

Observa algunas operaciones admitidas por las clases del módulo *datetime*.

```
from datetime import timedelta
from datetime import date
from datetime import datetime

delta = timedelta(weeks=2, days=2, hours=2)
print(delta)

delta2 = delta * 2
print(delta2)

d = date(2019, 10, 4) + delta2
print(d)

dt = datetime(2019, 10, 4, 14, 53) + delta2
print(dt)
```

Resultado:

```
16 days, 2:00:00
32 days, 4:00:00
2019-11-05
2019-11-05 18:53:00
```

El objeto *timedelta* se puede multiplicar por un número entero. En nuestro ejemplo, multiplicamos el objeto que representa 16 días y 2 horas por 2. Como resultado, recibimos un objeto *timedelta* que representa 32 días y 4 horas.

Toma en cuenta que tanto los días como las horas se han multiplicado por 2. Otra operación interesante usando el objeto *timedelta* es la suma. En el ejemplo, hemos sumado el objeto *timedelta* a los objetos date y *datetime*.

Como resultado de estas operaciones, recibimos objetos date y *datetime* incrementados en días y horas almacenados en el objeto *timedelta*.

La operación de multiplicación presentada te permite aumentar rápidamente el valor del objeto timedelta, mientras que la multiplicación también puede ayudar a obtener una fecha en el futuro.

Por supuesto, las clases *timedelta*, *date* y *datetime* admiten muchas más operaciones. Te recomendamos que

te familiarices con ellos en la documentación.

Puntos Clave

1. Para crear un objeto *date*, debes pasar los argumentos de año, mes y día de la siguiente manera:

```
from datetime import date

my_date = date(2020, 9, 29)
print("Año:", my_date.year) # Año: 2020
print("Mes:", my_date.month) # Mes: 9
print("Día:", my_date.day) # Día: 29
```

El objeto *date* tiene tres atributos (de solo lectura): año, mes y día.

2. El método today devuelve un objeto de fecha que representa la fecha local actual:

```
from datetime import date
print("Hoy:", date.today()) # Muestra: Hoy: 2020-09-29
```

3. En Unix, la marca de tiempo expresa el número de segundos desde el 1 de Enero de 1970 a las 00:00:00 (UTC). Esta fecha se llama la «época de Unix», porque ahí comenzó el conteo del tiempo en los sistemas Unix. La marca de tiempo es en realidad la diferencia entre una fecha en particular (incluida la hora) y el 1 de Enero de 1970, 00:00:00 (UTC), expresada en segundos. Para crear un objeto de fecha a partir de una marca de tiempo, debemos pasar una marca de tiempo Unix al método fromtimestamp:

```
from datetime import date
import time

timestamp = time.time()
d = date.fromtimestamp(timestamp)
```

Nota: La función time devuelve el número de segundos desde el 1 de Enero de 1970 hasta el momento actual en forma de número punto flotante.

4. El constructor de la clase time acepta seís argumentos (hour, minute, second, microsecond, tzinfo, y fold). Cada uno de estos argumentos es opcional.

```
from datetime import time

t = time(13, 22, 20)

print("Hora:", t.hour) # Hora: 13
print("Minuto:", t.minute) # Minuto: 22
print("Segundo:", t.second) # Segundo: 20
```

5. El módulo time contiene la función sleep, que suspende la ejecución del programa durante un número determinado de segundos, por ejemplo:

```
import time

time.sleep(10)
print("¡Hola mundo!") # Este texto se mostrará después de 10 segundos.
```

6. En el módulo datetime, la fecha y la hora se pueden representar como objetos separados o como un solo objeto. La clase que combina fecha y hora se llama datetime. Todos los argumentos pasados al constructor van a atributos de clase de solo lectura. Son year, month, day, hour, minute, second, microsecond, tzinfo, y fold:

```
from datetime import datetime

dt = datetime(2020, 9, 29, 13, 51)
print("Fecha y Hora:", dt) # Muestra: Fecha y Hora: 2020-09-29 13:51:00
```

7. El método strftime toma solo un argumento en forma de cadena que especifica un formato que puede constar de directivas. Una directiva es una cadena que consta del carácter % (porcentaje) y una letra minúscula o mayúscula. A continuación se muestran algunas directivas útiles:

- %Y: devuelve el año con el siglo como número decimal.
- %m: devuelve el mes como un número decimal con relleno de ceros.
- %d: devuelve el día como un número decimal con relleno de ceros.
- %H: devuelve la hora como un número decimal con relleno de ceros.
- %M: devuelve el minuto como un número decimal con relleno de ceros.
- %S: devuelve el segundo como un número decimal con relleno de ceros.

Ejemplo:

```
from datetime import date

d = date(2020, 9, 29)
print(d.strftime('%Y/%m/%d')) # Muestra: 2020/09/29
```

8. Es posible realizar cálculos en los objetos date y datetime, por ejemplo:

```
from datetime import date

d1 = date(2020, 11, 4)
d2 = date(2019, 11, 4)

d = d1 - d2
print(d) # Muestra: 366 days, 0:00:00.
print(d * 2) # Muestra: 732 days, 0:00:00.
```

El resultado de la resta se devuelve como un objeto timedelta que expresa la diferencia en días entre las dos fechas en el ejemplo anterior.

Toma en cuenta que también se muestra la diferencia en horas, minutos y segundos. El objeto timedelta se puede utilizar para realizar más cálculos (por ejemplo, puedes multiplicarlo por 2).

From:
https://matebcn.eu/ - **miguel angel torres egea**

Permanent link:
https://matebcn.eu/info:cursos:netacad:python:pe2m4:biblioteca:datetime

Last update: **07/07/2022 17:25**