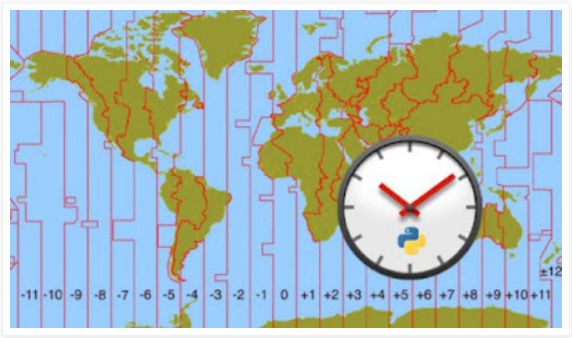
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

domingo, 5 de marzo de 2017

El módulo time



El módulo **time** de la biblioteca estándar de Python proporciona un conjunto de funciones para trabajar con fechas y/o horas. Además de estás funciones hay otras relacionadas en los módulos **datetime** y **calendar** que conviene conocer.

En el módulo **time** hay funciones para obtener la fecha y/o hora de distintos tipos de relojes (incluido el reloj que ofrece el tiempo civil); para obtener información relacionada con nuestro huso horario; para convertir fechas y/o horas entre distintos tipos; para validar y aplicar formatos de fechas y/o horas y para detener durante un tiempo la ejecución de un programa.

Una fecha y/o hora específica se puede expresar de tres modos diferentes:

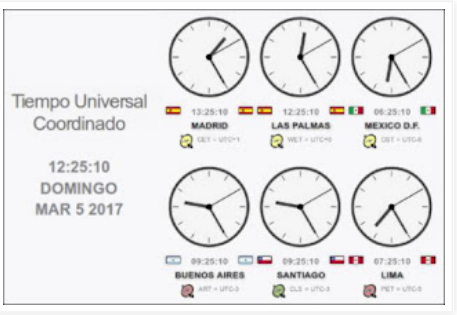
- como un número (float),
- como una cadena
- y como un objeto struct_time (es una tupla a medida (namedtuple) con nueve valores enteros).

Por ejemplo, la fecha-hora 4-3-2017 19:10:01 representada en los tipos anteriores:

- como un número: 1488651001.7188754 segundos
- como una cadena: "Sat Mar 4 19:10:01 2017"
- y como un objeto struct_time: time.struct_time(tm_year=2017, tm_mon=3, tm_mday=4, tm_hour=18, tm_min=10, tm_sec=1, tm_wday=5, tm_yday=63, tm_isdst=0)

Algunas funciones del módulo **time** operan sólo en un rango temporal denominado **época** que comienza el día 1 de enero de 1970 a las 0 horas y que finaliza en el año 2038 (en un sistema de 32 bit).

El tiempo civil se puede expresar como **tiempo local**, o bien, como tiempo **UTC** que es el **Tiempo Universal Coordinado** para/por todos los países. La diferencia entre ambos tiempos depende del punto geográfico (que se corresponde con un huso horario de 24 existentes) y de los ajustes que algunos países aplican en el horario de verano.



Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

El **tiempo UTC** se mantiene utilizando unos 400 relojes atómicos altamente precisos combinados con el tiempo de la rotación de la Tierra (Tiempo Solar).

Normalmente, para mantener la hora exacta un sistema informático sincroniza su reloj (el que mantiene la hora civil) con un servidor de tiempo dedicado de Internet que hace lo propio con alguno de los relojes atómicos que hay repartidos por todo el planeta.

Todos los husos horarios se definen en relación al tiempo UTC cuyo huso horario se sitúa centrado sobre el **Meridiano Cero** o **Meridiano de Greenwich**. Como la Tierra gira de oeste a este, al pasar de un huso horario al siguiente en dirección este se suma una hora; y si es en dirección oeste se resta una hora, hasta alcanzar el meridiano situado a 180º que es el que marca el cambio de día.

Funciones para obtener y convertir tiempo (civil)

Obtener tiempo local (fecha-hora) en segundos: time()

```
import time
tiempo_segundos = time.time()
print(tiempo_segundos)  # 1488651001.7188754 segundos

# El valor obtenido se corresponde con los segundos
# que han transcurrido desde el comienzo de la
# época: 1 de enero de 1970, 0 horas
```

Convertir a cadena tiempo en segundos: ctime()

```
tiempo_cadena = time.ctime(tiempo_segundos) # 1488651001.7188754 seg
print(tiempo_cadena)  # Sat Mar  4 19:10:01 2017

# Operando con tiempos

tiempo_segundos = 1488651001.7188754
tiempo_segundos = tiempo_segundos + 86400  # Suma 1 día
print(time.ctime(semana))  # Sun Mar  5 19:10:01 2017
```

Obtener tiempo UTC a partir de segundos: gmtime()

```
tiempo_st = time.gmtime(tiempo_segundos)
print(tiempo_st)

# time.struct_time(tm_year=2017, tm_mon=3, tm_mday=4,
#                  tm_hour=18, tm_min=10, tm_sec=1,
#                  tm_wday=5, tm_yday=63, tm_isdst=0)

# El valor devuelto es un objeto struct_time con
# los siguientes atributos:

# - tm_year: año
# - tm_mon: mes
# - tm_mday: día del mes
# - tm_hour: hora
# - tm_min: minutos
# - tm_sec: segundos
# - tm_wday: día de la semana [0, 6]
# - tm_yday: día del año [1, 366]
# - tm_isdst: horario de verano: 0 (no vigente),
#                                1 (vigente) y
#                               -1 (desconocido)

# En este caso por tratarse de la zona horaria de
# España (península) el tiempo devuelto en UTC
# tiene una hora menos que el tiempo local. Con el
# ajuste del horario de verano esa diferencia se
# amplia una hora más.

# Para acceder a los valores de los atributos de
# un objeto struct_time:

print(tiempo.st_year)  # 2017
print(tiempo.st_mon)  # 3
print(tiempo.st_hour)  # 18

# El módulo calendar tiene la función timegm() que
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

marzo 2017 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
# es la inversa de gmtime().

# Si el argumento es 0 segundo, el valor que se
# obtiene representa el momento de comienzo de la
# época: 1 de enero de 1970, 0 horas.

print(time.gmtime(0))

# time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1,
#                  tm_hour=0, tm_min=0, tm_sec=0,
#                  tm_wday=3, tm_yday=1, tm_isdst=0)
```

Obtener tiempo local como objeto struct_time: localtime()

```
# El tiempo devuelto por localtime() es equivalente
# a time() pero expresado como un objeto struct_time:

print('time():', time.ctime(time.time()),
      '\nlocaltime():', time.localtime())

# time(): Sat Mar  4 20:03:25 2017
# localtime(): time.struct_time(tm_year=2017, tm_mon=3, tm_mday=4,
#                               tm_hour=20, tm_min=3, tm_sec=25,
#                               tm_wday=5, tm_yday=63, tm_isdst=0)
```

Convertir a cadena un objeto struct_time: asctime()

```
tiempo_st = time.localtime()
print(time.asctime(tiempo_st))  # Sat Mar  4 20:22:13 2017
```

Convertir a segundos un objeto struct_time: mktime()

```
tiempo_st = time.localtime()
print(time.mktime(tiempo_st))  #  1488655494.0
```

Funciones para validar y aplicar formatos

Validar tiempo expresado como cadena: strptime()

```
# El tiempo se expresa como una cadena en el
# formato elegido y el valor que devuelve
# strptime() es un objeto struct_time:

import time
tiempo_st = time.strptime("4 Mar 2017", "%d %b %Y")
print(tiempo_st)

# time.struct_time(tm_year=2017, tm_mon=3, tm_mday=4,
#                  tm_hour=0, tm_min=0, tm_sec=0,
#                  tm_wday=5, tm_yday=63, tm_isdst=-1)


# Validando una fecha con el formato: dd-mm-aaaa
# Se produce una excepción porque la fecha es incorrecta

try:
    tiempo_st = time.strptime("34-03-2017", "%d-%m-%Y")

except:
    print('Fecha incorrecta')

# El formato es opcional. Si no se indica se
# utilizará: '%a %b %d %H:%M:%S %Y'
```

Convertir y formatear un objeto struct_time: strftime()

```
# El tiempo se expresa como objeto struct_time y
# el valor que devuelve la función strftime() es
# una cadena con el formato elegido:

tiempo_cadena = time.strftime("%d-%m-%Y %H:%M", time.gmtime())
print(tiempo_cadena)  #  04-03-2017 20:41
```

Funciones y variables para obtener información del huso horario

Los valores que se obtienen en los ejemplos siguientes son del huso horario de España (península). En las Islas Canarias los relojes marcan una hora menos que en la Península, es decir, el Tiempo UTC y el Tiempo Local son coincidentes con el horario de invierno vigente.

(Abreviaturas de husos horarios - Lista mundial)

```
# Obtener desplazamiento del huso local en verano.
# Diferencia en segundos entre Tiempo UTC y Tiempo
# Local con el horario de verano:

time.altzone  # -7200 segundos (-2 horas)

# Obtener desplazamiento del huso local.
# Diferencia en segundos entre Tiempo UTC y Tiempo
# Local con el horario local:

time.timezone()  # -3600 segundos (-1 hora)

# Obtener huso horario local ordinario y/o de verano
# Devuelve tupla con uno o dos cadenas dependiendo de
# si hay ajuste en el horario de verano.

time.tzname  # En España:('CET', 'CEST')

# La función time.tzset() permite cambiar el huso
# horario de referencia

# Conocer si en el huso horaria actual se aplica ajuste en
# horario de verano: (1 , 0, -1)

time.daylight  # 1 (vigente)
```

Funciones para obtener tiempo del sistema

Un equipo informático utiliza varios relojes para su funcionamiento y un programa Python con algunas funciones del módulo **time** puede obtener y utilizar el tiempo que marcan en un momento dado.

```
# Obtener tiempo en segundos de reloj monotónico
# Este tipo de relojes no pueden retroceder en el
# tiempo: siempre una llamada posterior a la función
# obtendrá un valor superior que una anterior.

time.monotonic()  # 5843.732800711
time.monotonic()  # 5847.289432529

# Obtener tiempo en segundos del contador de rendimiento
# El valor que se obtiene en este tipo de relojes es
# muy preciso. Se suele utilizar para medir tiempos
# cortos.

time.perf_counter()  # 6134.798769373

# Obtener tiempo de procesamiento en segundos
# Se obtiene sumando el tiempo de CPU y del proceso actual
# del usuario.

time.process_time()  # 8.249604247

# Obtener información sobre un tipo de reloj:
# 'monotonic', 'perf_counter', 'process_time' y 'time'

time.get_clock_info('time')

# namespace(adjustable=True,
#           implementation='clock_gettime(CLOCK_REALTIME)',
```

```
#           monotonic=False,
#           resolution=1e-09)
```

Función para detener un programa: sleep()

```
# La función sleep() permite detener la ejecución
# de un programa durante un tiempo.

# Se detiene cinco veces con un tiempo de parada
# que va aumentando cada vez un segundo:

for segundos in range(1,6):
    print('Detenido durante', segundos, 'segundos')
    time.sleep(segundos)
print('Proceso finalizado')
```

Relacionado:

- Operaciones con fechas y horas (datetime)

Ir al índice del tutorial de Python

Publicado por Pherkad en 11:42

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua