Fecha y hora en internet

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Abril de 2016

datetime

El lenguaje python tiene un tipo de datos muy conveniente para el manejo de fechas: datetime

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import datetime
def main():
    dt=datetime.datetime.now()
    print dt # 2016-03-05 17:39:15.900732
    print dt.year, # 2015
    print dt.month, # 3
    print dt.day, # 5
    print dt.hour, # 17
    print dt.minute, # 39
    print dt.second, # 15
    print dt.microsecond # 900732
if __name__ == "__main__":
    main()
```

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import datetime
def main():
    dt1=datetime.datetime(2016,1,1,0,0,0)
    print dt1  # 2016-01-01 00:00:00

    dt2=datetime.datetime.now()
    print dt2  # 2016-03-05 17:51:45.604391

    diferencia= dt2-dt1
    print diferencia  # 64 days, 17:51:45.604391
    print diferencia.total_seconds() # 5593905.60439

if __name__ == "__main__":
    main()
```

timestamp

El tipo de datos datetime es propio del lenguaje python. En otros lenguajes y en las librerías de Unix y de Windows lo habitual es usar *timestamp*, que es el número de segundos transcurridos desde un instante inicial denominado *epoch*

- En Unix el *epoch* es el 1 de enero de 1970 a las 00:00 UTC
- En Windows el *epoch* es el 1 de enero de 1601 a las 00:00 UTC

El *Unix time* es el *timestamp* basado en el *epoch* Unix

Timestamp es el formato en que se almacenan y procesan las fechas, cuando se tienen que presentar a las personas se emplea algún formato más legible

- El problema es que los formatos humanos para representar fechas son muy heterogéneos
  - Dia, mes y año en Europa, parte de Asia, norte de Africa, América Central, América del Sur
  - Mes, día y año en Estados Unidos
  - Años, mes y día en China
- Una alternativa normalizada es el ISO8601, similar al formato chino
  2016-03-05T06:24:57+00:00

- En python disponemos de la librería *time* para trabajar con *timestamps*
- Pero *datetime* normalmente resulta más conveniente. Cuando nuestro programa tenga que aceptar y generar *timestamps* es recomendable
  - En la entrada, convertir el *timestamp* en *datetime*
  - Procesar el *datetime*
  - En la salida, generar o bien un *timestamp* o bien algún formato legible para los humanos, según nuestros requisitos

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import time,datetime
def main():
    # Obtenemos la hora actual, como timestamp
    ts=time.time()
    print ts    # 1457197258.88

    # Convertimos el timestamp a datetime naive
    dt=datetime.datetime.utcfromtimestamp(ts)
    print dt  # 2016-03-05 18:00:58.881041

    # Convertimos el datetime naive de nuevo a timestamp
    ts=time.mktime( dt.timetuple())
    print ts # 1457197258.0

if __name__ == "__main__":
    main()
```

Zonas horarias

En cualquier aplicación que funcione en internet es muy importante tener en cuenta la zona horaria

- Los sistemas operativos que usamos actualmente y la mayoría de los lenguajes de programación actuales son anteriores a internet
- Una aplicación tradicional suele ejecutarse en algún lugar del mundo en concreto, con lo que no es especialmente importante especificar la zona horaria. Suele sobreentender *hora local*
- En internet no existe la *hora local*. Es imprescindible indicar siempre la zona horaria
- En la actualidad, cualquier representación de una hora que no especifique claramente su zona horaria, es ambigüa y muy desaconsejable

- Las horas locales son muy complejas, es necesario tener en cuenta no solo el huso horario sino la presencia o ausencia del horario de verano (adelantar una hora a principios de la primavera y se retrasarla en otoño)
- Lo más recomendable es almacenar y procesar todas las fechas en UTC *Coordinated Universal Time*
  - Esta hora es casi idéntica a la antigua GMT, *Greenwich Mean Time*
- Cualquier programa debería usar siempre horas UTC, y convertirla a hora local solo inmediatamente antes de presentarla a un humano

- El tipo de datos *timestamp* no requiere indicar cuál es su zona horaria, se trata de los segundos transcurridos desde el epoch, y el epoch siempre es UTC
- Con el tipo *datetime* sí es necesario indicar zona horaria

pytz

- En python, el formato datetime tradicional no incluye información sobre la zona horaria
- Actualmente se le denomina *naive datetime* (datetime ingenuo)
- En un datetime naive, es necesario sobreentender si se trata de una hora local o una hora UTC
  - Obviamente, esto es muy problemático
- El soporte para zonas horarias en python no es muy bueno. Es recomendable usar la librería pytz, que no está incluida en la distribución estándar de python, es necesario instalarla
  `pip install pytz`

- El método strftime del tipo datetime admite una cadena de formato, con la que podemos indicar que el datetime se muestre indicando zona horaria

```
fmt = "%Y-%m-%d %H:%M:%S %Z%z"
print dt.strftime(fmt) # 2016-03-05 19:47:53 UTC+0000
```

  - Aunque si el datetime es naive, no mostrará ninguna zona

- Un objeto pytz contiene la localización de una zona horaria

```
madrid=pytz.timezone("Europe/Madrid")
```

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-
import datetime,pytz
def main():
    usa_eastern=pytz.timezone("US/Eastern")
    utc=pytz.utc
    fmt = "%Y-%m-%d %H:%M:%S %Z%z"

    dt=datetime.datetime(2016,7,4,tzinfo=usa_eastern)
    print dt.strftime(fmt) # 2016-07-04 00:00:00 EST-0500

    dt=datetime.datetime(2016,10,12,12,0, tzinfo=utc)
    print dt.strftime(fmt) # 2016-10-12 12:00:00 UTC+0000

if __name__ == "__main__":
    main()
```

- A partir de un datetime naive, podemos añadirle zona horaria con el método *localize* del objeto *pytz*

```
dt=madrid.localize(dt)
print dt.strftime(fmt) # 2016-03-05 20:47:53 CET+0100
```

- Cuando un datetime tiene zona horaria, podemos obtener otro datetime de diferente zona con el método *astimezone* del objeto *pytz*

```
dt=dt.astimezone(utc)
print dt.strftime(fmt) # 2016-03-05 19:47:53 UTC+0000
```

- Observa que *localize* es un método del *timezone* que recibe un *datetime*, mientras que *astimezone* es un método del *datetime* que recibe un *timezone*

```
import datetime,pytz
def main():
    madrid=pytz.timezone("Europe/Madrid")
    pekin=pytz.timezone("Asia/Shanghai")
    utc=pytz.utc

    dt=datetime.datetime.now()
    fmt = "%Y-%m-%d %H:%M:%S %Z%z"

    # dt naive, sin información de zona horaria
    print dt.strftime(fmt) # 2016-03-05 20:47:53

    # Añadimos zona horaria
    dt=madrid.localize(dt)
    print dt.strftime(fmt) # 2016-03-05 20:47:53 CET+0100

    # Cambiamos la zona horaria
    dt=dt.astimezone(pekin)
    print dt.strftime(fmt) # 2016-03-06 03:47:53 CST+0800

    dt=dt.astimezone(utc)
    print dt.strftime(fmt) # 2016-03-05 19:47:53 UTC+0000
```

Conversión datetime con zona en timestamp

A partir de un datetime, podemos necesitar convertirlo en un timestamp

- Si el datetime tiene zona horaria, ya no es aplicable *time.mktime()*

  time.mktime(dt.timetuple())

- Debemos usar el método *timegm()* del módulo *calendar*

```
import calendar

ts=calendar.timegm(dt.utctimetuple())
```

Lista completa de zonas horarias

```
#!/usr/bin/python -tt
import pytz
def main():
    for tz in pytz.all_timezones:
        print tz
if __name__ == "__main__":
    main()
```

Resultado:

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
[...]
US/Mountain
US/Pacific
US/Pacific-New
US/Samoa
UTC
Universal
W-SU
WET
Zulu
```