

Un'Introduzione alla Rappresentazione, Serializzazione e Gestione del Tempo nel Software

La maggior parte dei problemi nello sviluppo software solitamente nascono da una conoscenza scarsa e inconsistente del dominio in questione. Un argomento apparentemente semplice come la rappresentazione, serializzazione e gestione del tempo può facilmente causare una serie di problemi sia al programmatore neofita che a quello esperto.

In questo post, vedremo che non c'è bisogno di essere un Signore del Tempo per afferrare i pochissimi concetti semplici necessari per non incappare nell'inferno della gestione del tempo.



Rappresentazione

Una domanda semplice come *"Che ore sono?"* presuppone una serie di sfumature contestuali che sono ovvie per il cervello umano, ma diventano assolutamente prive di senso per un computer.

Per esempio, se mi stessi chiedendo che ore sono proprio ora, potrei dire: *"Sono le 15:39"* e, se fossi un collega nel mio ufficio, questa sarebbe un'informazione sufficiente per dedurre che sono

le 15:39 CEST. Questo perché saresti già in possesso di alcuni bit di informazioni contestuali importanti come

- è pomeriggio perché abbiamo già pranzato
- siamo a Roma, quindi il nostro fuso orario è l'Ora dell'Europa Centrale (CET) o l'Ora Estiva dell'Europa Centrale (CEST)
- siamo passati all'ora legale alcune settimane fa, quindi il fuso orario corrente deve essere l'Ora Estiva dell'Europa Centrale

15:39 è una rappresentazione conveniente del tempo finché siamo in possesso dei bit contestuali. Per rappresentare il tempo in modo universale, dovrete avere un'idea di cosa siano UTC e i fusi orari.

Vi prego di non confondere UTC con GMT: anche se il loro orario coincide, sono *due cose diverse*: uno è uno standard universale mentre l'altro è un fuso orario. Quando qualcuno dice di usare GMT, a meno che quella persona non abbia un divertente accento scozzese, quello che intende veramente è UTC.

Come radioamatore, ho contatti con persone da tutto il mondo. Ogni operatore è tenuto a mantenere un registro dei suoi contatti, e di solito ci scambiamo le cosiddette cartoline QSL, che sono una conferma scritta del contatto. Ovviamente una cartolina QSL deve riportare l'orario esatto del contatto radio e per convenzione è in UTC. So che quando ricevo una cartolina QSL da qualsiasi collega radioamatore, non importa dove sia localizzato in tutto il mondo, posso consultare il contatto nel mio registro e le informazioni di data e ora corrisponderanno, dato che stiamo entrambi aderendo allo stesso standard: UTC.

Ora, supponiamo di dover programmare una chat Skype con un collega sviluppatore negli Stati Uniti. Potrei scrivergli una email e dire qualcosa come "*ci vediamo il 2/3*". In Italia, questo sarebbe il secondo giorno del mese di marzo, ma per una persona statunitense, questo sarebbe il terzo giorno del mese di febbraio. Come potete vedere, la nostra chat non avverrà mai.

Questi sono solo alcuni esempi del tipo di problemi che potrebbero sorgere quando si rappresentano informazioni di data e ora. Fortunatamente, c'è una soluzione agli enigmi della rappresentazione, ovvero lo standard ISO 8601 o, meglio ancora, RFC 3339.

Solo per darvi un esempio, in RFC 3339, 1994-11-05T08:15:30-05:00 corrisponde al 5 novembre 1994, 8:15:30 am, Ora Standard dell'Est degli Stati Uniti. 1994-11-05T13:15:30Z corrisponde allo stesso istante (la Z sta per UTC). Stesso istante, rappresentazioni diverse.

RFC 3339 ha anche il bel effetto collaterale di fornire ordinamento naturale in sistemi che utilizzano l'ordine lessicografico (come i filesystem) perché l'informazione è organizzata dalla più alla meno significativa, ovvero anno, mese, giorno, ora, minuto, secondo, frazione di secondo¹.

Anche se state gestendo solo orari locali nel vostro software, dovrete sapere che, a meno che non visualizzate anche il fuso orario, non potete mai essere sicuri dell'orario. Non riesco a ricordare

¹assumendo che lo stesso fuso orario sia usato ovunque

quante volte uno sviluppatore mi ha chiesto di *sistemare l'orario* sul server, solo per scoprire che il suo software stava stampando l'orario in UTC.

Al momento della visualizzazione, è normale gestire rappresentazioni parziali del tempo perché l'esperienza utente lo richiede. Assicuratevi solo, durante il debugging, di stampare l'intero set di informazioni, incluso il fuso orario, altrimenti non potrete mai essere sicuri che quello che state guardando sia quello che pensate veramente che sia.

Anche se un dato momento nel tempo è immutabile, c'è un numero arbitrario di modi per esprimerlo. E non abbiamo nemmeno parlato dei calendari giuliano o indiano o di cose come esprimere le durate!

Lasciatemi riassumere alcuni punti chiave da portare a casa finora:

- imparate a conoscere i fusi orari e UTC
- non confondete UTC e GMT
- RFC 3339 e ISO 8601 sono vostri amici
- stampate sempre il fuso orario durante il debugging



Serializzazione

Parlando di software, la serializzazione è un processo dove prendete lo stato di un oggetto e lo specificate in modo tale che possa essere successivamente interamente ricostruito, esattamente come l'originale, utilizzando le informazioni esplicitate (serializzate). Pensate a un file xml o json:

```
{
  "person": {
    "name": "Mirko",
    "surname": "Caserta",
    "class": "nerd"
  }
}
```

```
}  
}
```

Questa è la forma serializzata di una particolare istanza immaginaria della classe `persona`.

Nel mondo binario dei computer, il tempo è solitamente serializzato e memorizzato utilizzando la convenzione del tempo Unix. Mentre sto scrivendo questo, il mio tempo Unix è 1366191727. Ovvero: 1366191727 secondi sono passati dal 1° gennaio 1970 alle 00:00 UTC. Non è questo un modo piuttosto intelligente, consistente e compatto di rappresentare una pletora di informazioni, come 17 aprile 2013 @ 11:42:07am CEST?

Il tempo Unix è solo un'altra rappresentazione arbitraria di un dato momento nel tempo, anche se non molto leggibile per gli umani. Ma potete prendere quel numero, scriverlo su un pezzo di carta, attaccarlo a un piccione viaggiatore, e il vostro destinatario sarebbe in grado di decifrare il vostro messaggio vitale semplicemente rivolgendosi a Internet e visitando un sito come unixtimestamp.com o currentmillis.com.

Se siete dei drogati della riga di comando come me, sui sistemi Linux potete usare:

```
$ date -d @1366191727  
Wed Apr 17 11:42:07 CEST 2013
```

Tuttavia, sui sistemi derivati da BSD come Mac OS X, `date -d` non funziona quindi dovete usare invece:

```
$ date -r 1366191727  
Wed Apr 17 11:42:07 CEST 2013
```

Proprio come potete scrivere quel numero su un pezzo di carta e successivamente riportare in vita l'istante completo, potete memorizzarlo in un file o in una riga del vostro RDBMS preferito. Anche se potreste voler parlare al vostro RDBMS usando un driver appropriato e passargli una semplice istanza di data; il vostro driver si occuperà poi della conversione al formato di serializzazione del database sottostante per le istanze di tempo native.

Memorizzando il tempo usando un formato nativo, ottenete gratuitamente le ottime funzionalità di formattazione, ordinamento, interrogazione, ecc. del vostro RDBMS per il tempo, quindi potreste voler pensarci due volte prima di memorizzare semplici timestamp Unix in, diciamo, Oracle.

Assicuratevi solo di sapere a quale fuso orario si riferisce il vostro timestamp Unix, o potreste confondervi successivamente al momento della deserializzazione. Per default, un timestamp Unix è in UTC. Se usate le librerie del vostro sistema, dovrete essere a posto.

Quando lavorate con database, usate i tipi di dati più appropriati. Per esempio in Oracle, ci sono quattro tipi di dati diversi: `DATE`, `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE` e `TIMESTAMP WITH LOCAL TIME ZONE`. Inoltre, i database solitamente hanno un concetto di fuso orario del database e fuso orario della sessione, quindi assicuratevi di capire come il vostro database specifico li sta usando. Un utente che apre una sessione con fuso orario A vedrà valori diversi rispetto a un utente che si connette con fuso orario B.

ISO 8601 è anche un favorito per la serializzazione. Infatti, è usato nello standard XML Schema. La maggior parte dei framework xml è nativamente in grado di serializzare e deserializzare avanti e indietro da `xs:date`, `xs:time` e `xs:dateTime` al formato nativo del vostro linguaggio di programmazione (e viceversa). Lo stesso vale per json. Fate solo attenzione quando gestite rappresentazioni parziali: per esempio, se omettete il fuso orario, assicuratevi di concordare preventivamente su uno predefinito con la vostra controparte comunicante (solitamente UTC o il vostro fuso orario locale se siete entrambi nello stesso).

Gestione

Prima di tutto, se pensate di poter scrivere la vostra libreria software per la gestione del tempo, o anche scrivere una piccola routine che aggiunga o sottragga valori arbitrari dall'ora del giorno, permettetemi di mostrarvi il codice sorgente per le classi `java.util.Date` e `java.util.GregorianCalendar` del JDK 7, che pesano rispettivamente 1331 e 3179 righe di codice.

Ok, questi probabilmente non sono i migliori esempi di routine software che gestiscono il tempo, sono d'accordo. Ecco perché sono state scritte librerie Java come Joda Time. Infatti, Joda Time è diventata così popolare che ha dato vita a JSR-310 ed è ora parte del JDK 8.

L'uso di framework per il tempo popolari, ben progettati e implementati vi salverà la vita. Seriamente. Prendetevi il tempo per familiarizzare con l'API di vostra scelta.

Compiti Comuni per il Tempo in Java

Vediamo come tutto questo si traduce in codice java. Qualsiasi linguaggio sarà ovviamente diverso ma tutto quello che sto facendo qui dovrebbe essere possibile nel linguaggio che preferite.

Vi prego di non usare `java.util.Date` o `java.util.Calendar`. Non usiamo più quelle classi. La nuova API per il tempo è nel package `java.time`.

```
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;

class Main {
    public static void main(String[] args) {
        ZoneId systemDefault = ZoneId.systemDefault();
        System.out.println("systemDefault = " + systemDefault);

        long now = System.currentTimeMillis();
        System.out.println("now = " + now);

        LocalDate localDate = LocalDate.now();
        System.out.println("localDate = " + localDate);

        LocalDateTime localDateTime = LocalDateTime.now();
        System.out.println("localDateTime = " + localDateTime);

        LocalDateTime utc = LocalDateTime.now(ZoneId.of("UTC"));
        System.out.println("utc = " + utc);
    }
}
```

```

ZonedDateTime zonedDateTime1 = ZonedDateTime.now();
System.out.println("zonedDateTime1 = " + zonedDateTime1);

ZonedDateTime zonedDateTime2 = ZonedDateTime.now(ZoneId.of("UTC"));
System.out.println("zonedDateTime2 = " + zonedDateTime2);

String iso8601 = zonedDateTime2.format(DateTimeFormatter.ISO_INSTANT);
System.out.println("iso8601 = " + iso8601);

ZonedDateTime zonedDateTime3 = zonedDateTime2.plus(Duration.ofDays(7));
System.out.println("zonedDateTime3 = " + zonedDateTime3);

Instant nowAsInstant = Instant.ofEpochMilli(now);
System.out.println("nowAsInstant = " + nowAsInstant);

ZonedDateTime nowAsInstantInRome = nowAsInstant.atZone(ZoneId.of("Europe/
Rome"));
System.out.println("nowAsInstantInRome = " + nowAsInstantInRome);

LocalDateTime romeLocalTime = nowAsInstantInRome.toLocalDateTime();
System.out.println("romeLocalTime = " + romeLocalTime);

LocalDate localDateInRome = nowAsInstantInRome.toLocalDate();
System.out.println("localDateInRome = " + localDateInRome);

LocalTime localTimeInRome = nowAsInstantInRome.toLocalTime();
System.out.println("localTimeInRome = " + localTimeInRome);

String shortTimeInRome =
nowAsInstantInRome.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.SHORT));
System.out.println("shortTimeInRome = " + shortTimeInRome);

String evenShorterTimeInRome =
nowAsInstantInRome.format(DateTimeFormatter.ofPattern("HH:mm"));
System.out.println("evenShorterTimeInRome = " + evenShorterTimeInRome);
}
}

```

Se eseguite questo codice con `java Main.java`, dovrete vedere qualcosa di simile:

```

systemDefault = Europe/Rome
now = 1753718631998
localDate = 2025-07-28
localDateTime = 2025-07-28T10:03:51.999991
utc = 2025-07-28T16:03:52.000089
zonedDateTime1 = 2025-07-28T10:03:52.000532-06:00[Europe/Rome]
zonedDateTime2 = 2025-07-28T16:03:52.000620Z[UTC]
iso8601 = 2025-07-28T16:03:52.000620Z
zonedDateTime3 = 2025-08-04T16:03:52.000620Z[UTC]

```

```
nowAsInstant = 2025-07-28T16:03:51.998Z
nowAsInstantInRome = 2025-07-28T18:03:51.998+02:00[Europe/Rome]
romeLocalTime = 2025-07-28T18:03:51.998
localDateInRome = 2025-07-28
localTimeInRome = 18:03:51.998
shortTimeInRome = 6:03PM
evenShorterTimeInRome = 18:03
```

Test

In Java c'è una classe `Clock` che permette di collegare un'implementazione di orologio arbitrariamente configurabile per l'uso nell'API del tempo. Questo è particolarmente utile nei test unitari e nel debugging. Altri linguaggi dovrebbero avere una funzionalità equivalente.

<https://docs.oracle.com/javase/8/docs/api/java/time/Clock.html>

Risorse Aggiuntive

Ecco alcuni link utili che ho accumulato nel tempo:

- How to Think About Time in Programming
- UTC is enough for everyone... right?
- The Problem with Time & Timezones - Computerphile
- Falsehoods programmers believe about time
- Storing UTC is not a silver bullet
- The 5 laws of API dates and times
- Storing Date/Times in Databases
- 5 Levels of Handling Date and Time in Python
- Timezone Bullshit
- ISO 8601: the better date format
- A summary of the international standard date and time notation
- A Short History of the Modern Calendar
- Should We 'Heed the Science and Abolish Daylight Saving Time'?
- When a Calendar Defeated Russia in the 1908 Olympics
- Why does China Have Only One Time Zone?
- First day meme
- Glory to ISO8601 Subreddit
- Time.is
- How Ancient Romans Kept Time
- rtc: rk808: Compensate for Rockchip calendar deviation on November 31st
- Daylight saving time is 'not helpful' and has 'no upsides' experts say
- Neil deGrasse Tyson Reminds Us Daylight Saving Time is Ridiculous
- A "Day" Isn't What It Used To Be
- Why Time Zones Exist
- xkcd: Daylight Calendar
- xkcd: Edge Cake
- How does Britain know what time it is?

- Clockwork raises \$21M to keep server clocks in sync
- How to Fix Daylight Saving Time
- US Senate Unanimously Passes Bill to Make Daylight Saving Time Permanent
- The Daily WTF: Starting your Date
- Tech Giants Want To Banish the Leap Second To Stop Internet Crashes
- Meta calls for the death of the leap second
- Satellites Keep the World's Clocks on Time. What if They Fail?
- Stop using `utcnow` and `utcfromtimestamp`
- Dall'ora legale all'ora solare: i pro e contro del cambio orario e perché si parla di abolizione
- Deer-vehicle collisions spike when daylight saving time ends
- The Falling Dates
- Scientists Don't Want to Count Leap Seconds, so They're Going Away
- What time is it on the Moon?
- What time is it on the moon? Europe pushing for lunar time zone
- What time is it? A simple question with a complex answer. How computers synchronize time
- The Daylight Saving Time Mess Just Won't Go Away
- The best way to handle time zones in a Java web application
- Your Calendrical Fallacy Is...
- Greenland Solves the Daylight Saving Time Debate
- Time Zone and Currency Database in JDK
- (There Ought To Be A) Moonlight Saving Time
- We don't do DST at this company
- Daylight Savings Time be like
- List of 2024 leap day bugs
- California State Legislator Proposes Ending DST
- The science behind why people hate Daylight Saving Time so much
- Storing UTC is not a silver bullet
- JS Dates Are About to Be Fixed
- Researchers Figure Out How To Keep Clocks On the Earth, Moon In Sync 13
- NASA confirms it's developing the Moon's new time zone
- Storing time for human events
- Date and Time Mappings with Hibernate and JPA
- I Found the Dumbest Time Zone
- A Server for Matching Long/Lat to Timezone
- Ianto Cannon's clock graphics
- AlphaDec, a timezone-agonistic time format for humans, machines, and AI
- RFC 3339 vs ISO 8601
- How the tz database works
- A literary appreciation of the Olson/Zoneinfo/tz database
- Benjamin Franklin's Essay on Daylight Saving
- How England Colonized Time
- Cheatsheet & Examples: date

Meme



I Am Devloper
@iamdevloper

Elon Musk: I'm putting people on Mars!

Developers: Fantastic, more timezones to support.

2/9/18, 06:03

You're feeding me
an hour later
because of
Daylight Saving
Time?



How the fuck can I read a clock, John?





Friend: What happened?

**Me: I had to work with
timezones today**



**“For all I know, the universe started
on January 1st in 1970 exactly
midnight”**

- unsigned long



the stayer at home

@mountain_ghosts

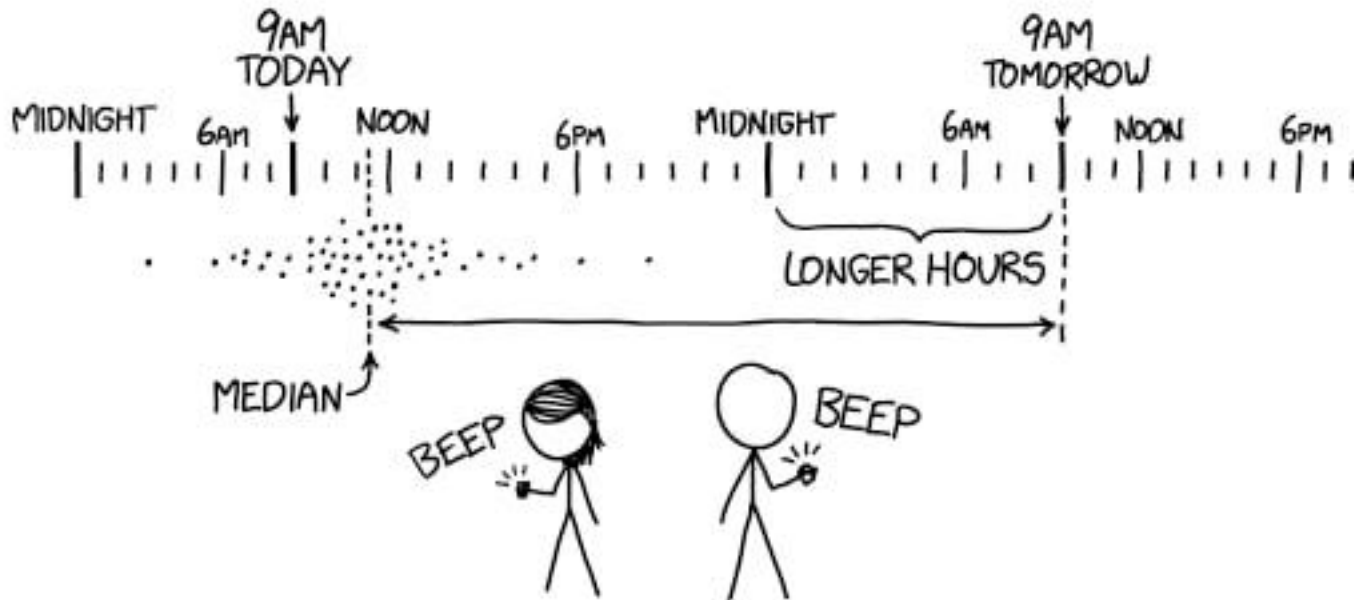
they put a millisecond clock with a 32-bit register, in a plane, and it overflows



Boeing 787s must be turned off and on every 51 days to prevent 'misleading data' being shown...
theregister.co.uk

PROPOSAL: CONSENSUS TIME

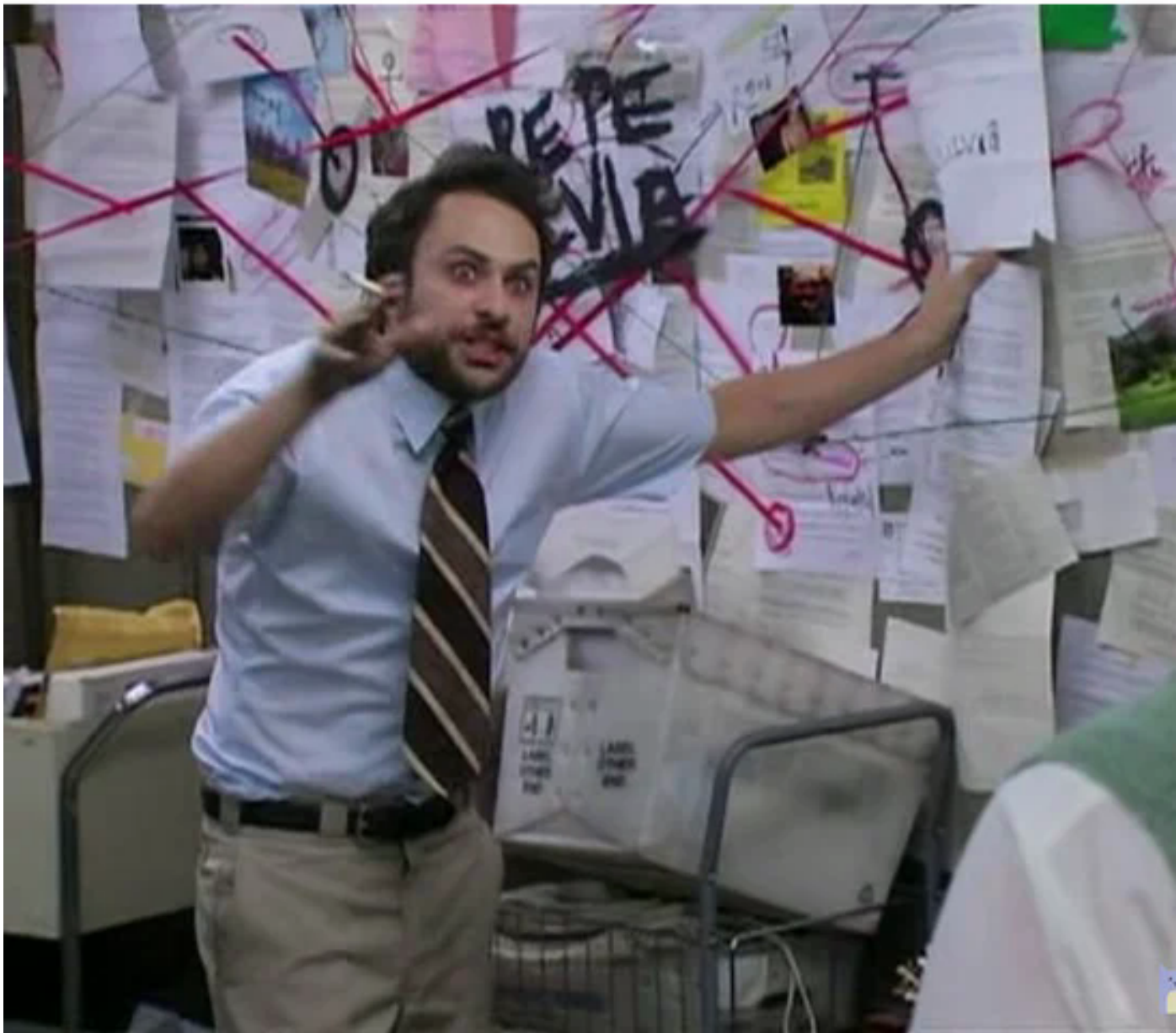
EVERY DAY, ANYONE IN THE TIME ZONE CAN PRESS A BUTTON WHEN THEY FEEL LIKE IT'S 9 AM. THE NEXT DAY, CLOCKS SLOW DOWN OR SPEED UP TO MATCH THE MEDIAN CHOICE FROM THE PREVIOUS DAY.



**DATE TIME LIBRARY
AUTHORS HAVING TO IMPLEMENT
YET ANOTHER TIMEZONE CHANGE**



**DEVELOPERS EXPLAINING WHY YOU HAVE
STORE DATE/TIME & TIMEZONE SEPARATELY
THEN MERGE IT ALL BACK TOGETHER JUST
TO CONVERT IT TO A NEW DATE/TIME THAT
MATCHES THE TIMEZONE FROM THE USER DEVICE**



Days since last
timezone issue

0

A GUIDE TO PUTTING YOUR CLOCKS



SMARTPHONE

Leave it alone,
it does its magic



OVEN

You'll need a
Masters in
Electronic
Engineering
or a hammer

**WHY CAN'T WE MOVE
THE CLOCKS AHEAD FROM
AFTERNOON AROUND 4:00**



01/01/2023





Terrible Maps @TerribleMaps@en.osm.town

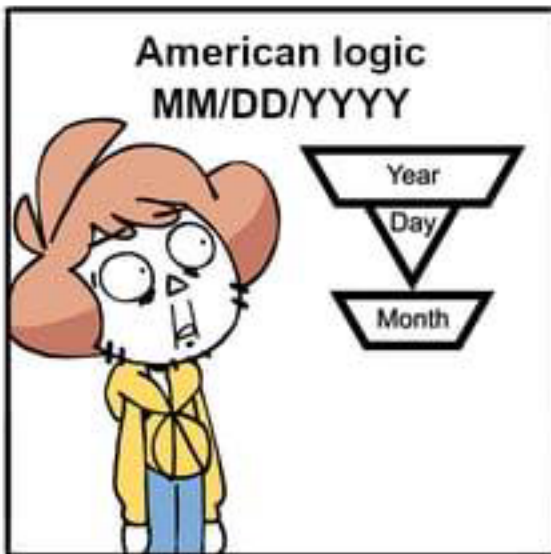
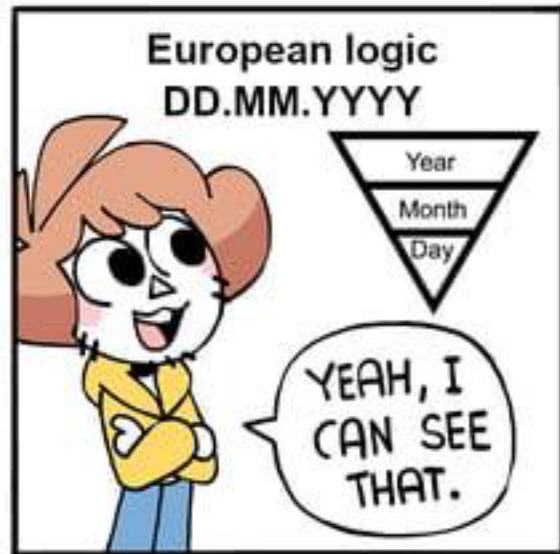
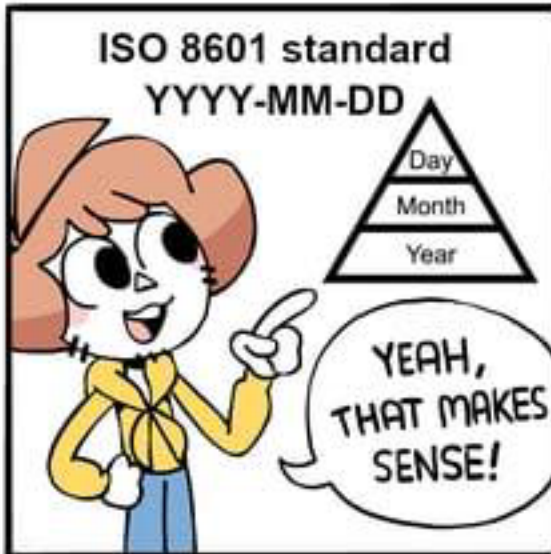
A comprehensive map of all countries that use
format



20

173

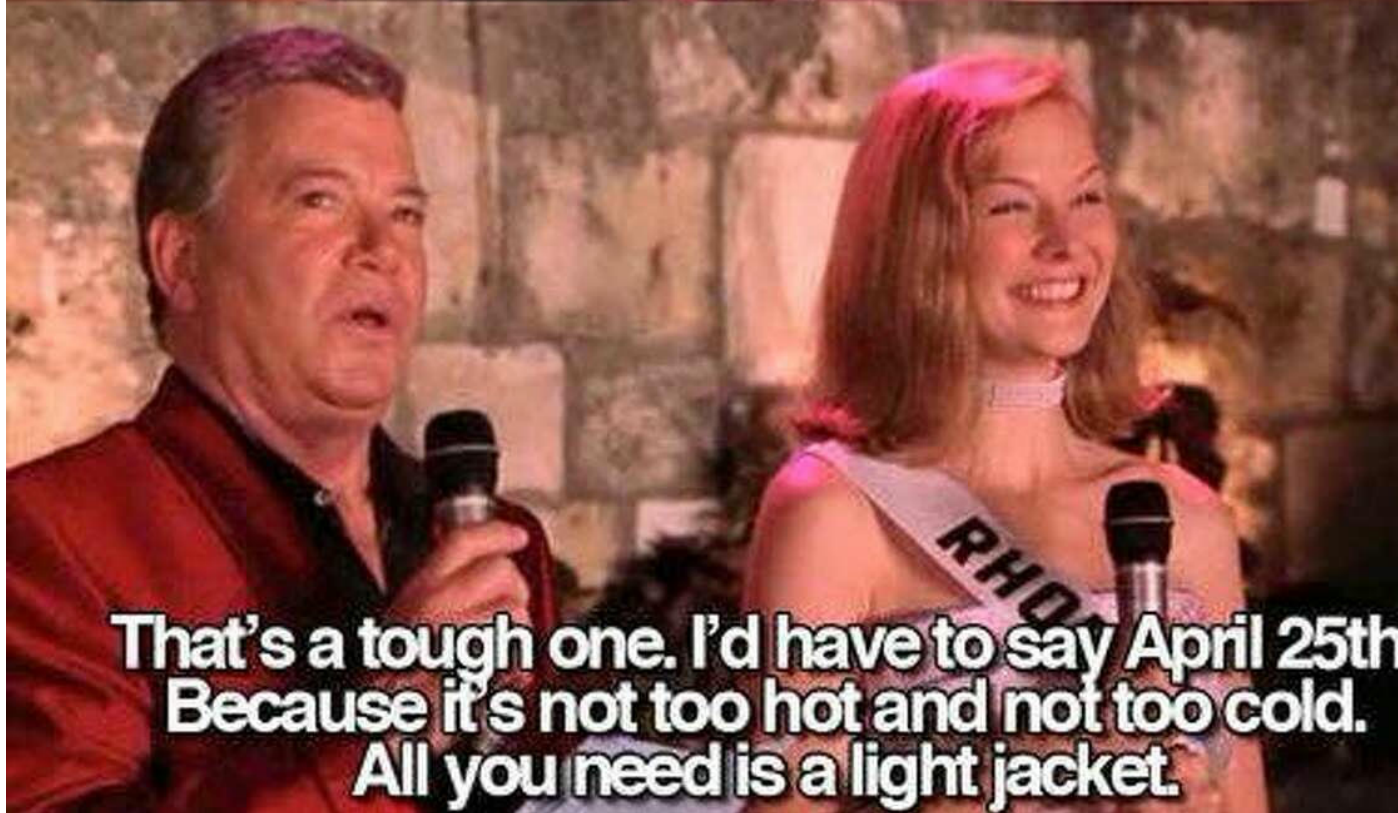
Date Formats



SHEN COMIX



Describe your perfect date.



**That's a tough one. I'd have to say April 25th
Because it's not too hot and not too cold.
All you need is a light jacket.**



kim

@KimmyMonte

live footage of daylight savings taking the sun away at 4pm

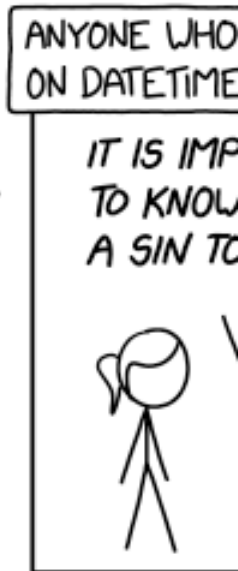
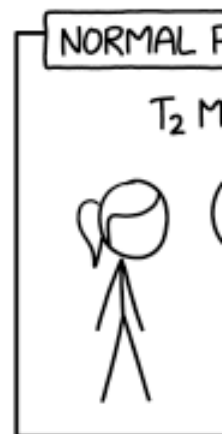
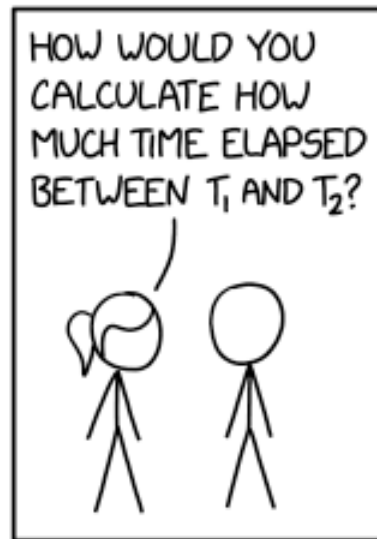


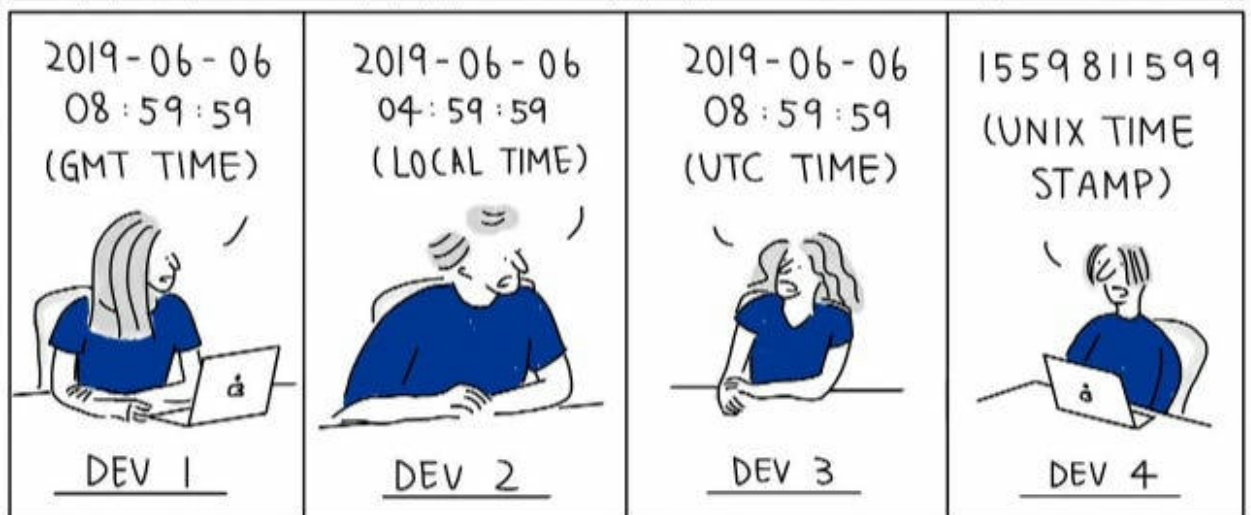
1:53 AM · 2023-11-05 · **9K** Views

30 Reposts **1** Quote **379** Likes **5** Bookmarks

Overnight workers watching t
from 1:59 to 1:00







Informazioni sul Documento

Questo documento è stato scaricato da <https://mirkocaserta.com/posts/time.it.pdf>

Generato il 2025-09-19 07:37:27 UTC