THE DATE-ING GAME

SWIFTCRAFT | FOLKESTONE, UK | MAY 2024

ELLEN SHAPIRO | MASTODON.SOCIAL/@DESIGNATEDNERD | PIXITEAPPS.COM


10


Organic




THURSDAY 4TH JULY 2024
BINDEPENDENCE DAY
TIME TO TAKE OUT THE TRASH

TIME SEEMS EASY

TIME SEEMS EASY

(SPOILER ALERT: IT IS NOT)

```
let now = Date()
```

60 * 60

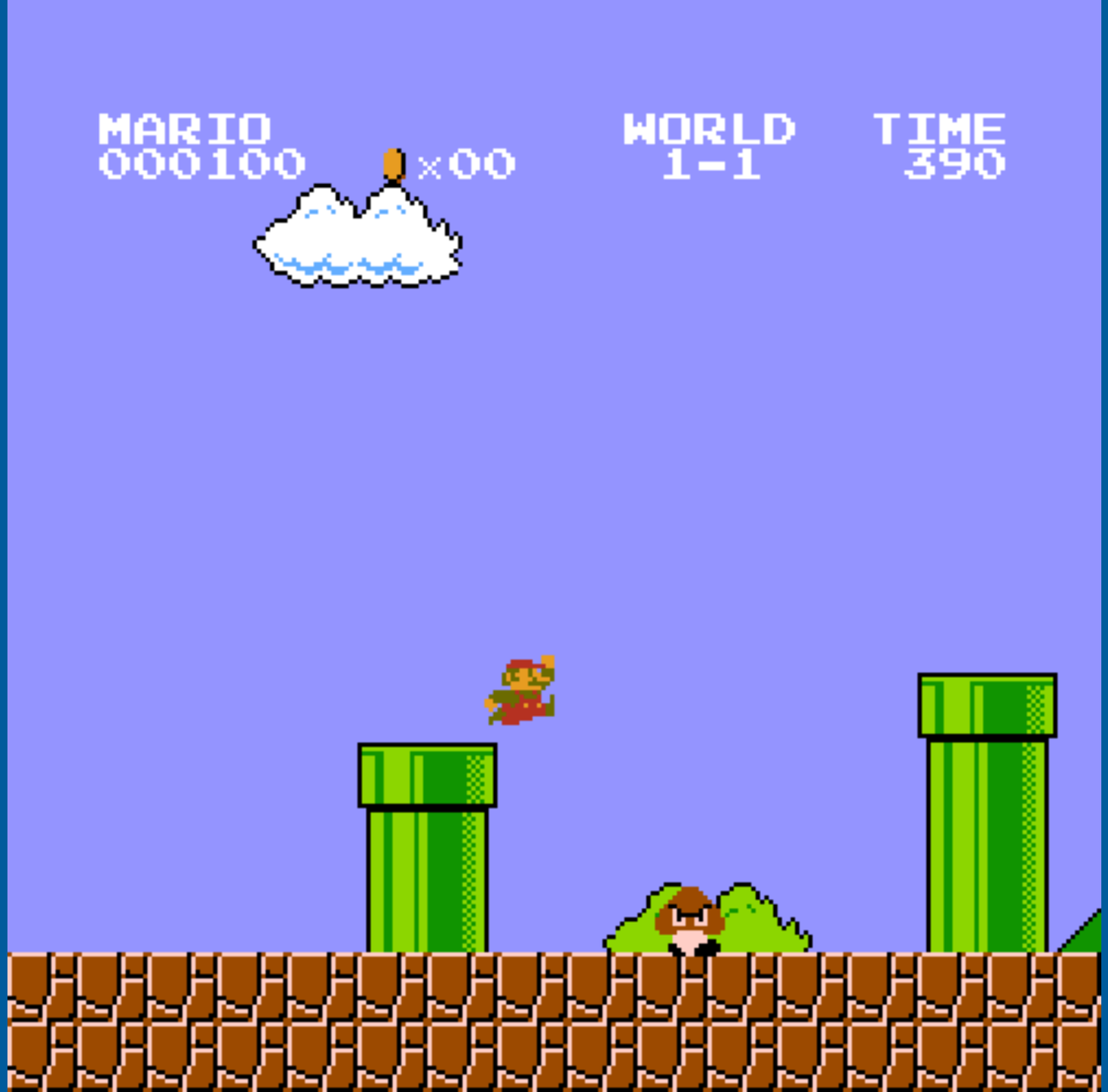
60 * 60 * 24

60 * 60 * 24 * 365

```
let now = Date.now
let oneYear = TimeInterval(60 * 60 * 24 * 365)

let oneYearLater = now.addingTimeInterval(oneYear)
print("Now: \(now).\nOne year from now: \(oneYearLater)")
```

```swift
let now = Date.now
let oneYear = TimeInterval(60 * 60 * 24 * 365)

let oneYearLater = now.addingTimeInterval(oneYear)
print("Now: \(now).\nOne year from now: \(oneYearLater)")
```

```
Now: 2024-01-07 22:59:11 +0000.
One year from now: 2025-01-06 22:59:11 +0000
```


MARIO
000100
×00
WORLD
1-1
TIME
390


Time is a flat circle

DATE

THE 🍎 DEFINITION

A SPECIFIC POINT IN TIME. INDEPENDENT OF ANY CALENDAR OR TIME ZONE.

MY DEFINITION

A FIXED POINT IN TIME

```
.timeIntervalSince1970
```

```
.timeIntervalSince1970
.timeIntervalSinceReferenceDate
```

.timeIntervalSince1970

00:00:00 UTC ON 1 JANUARY 1970

.timeIntervalSinceReferenceDate

`.timeIntervalSince1970`

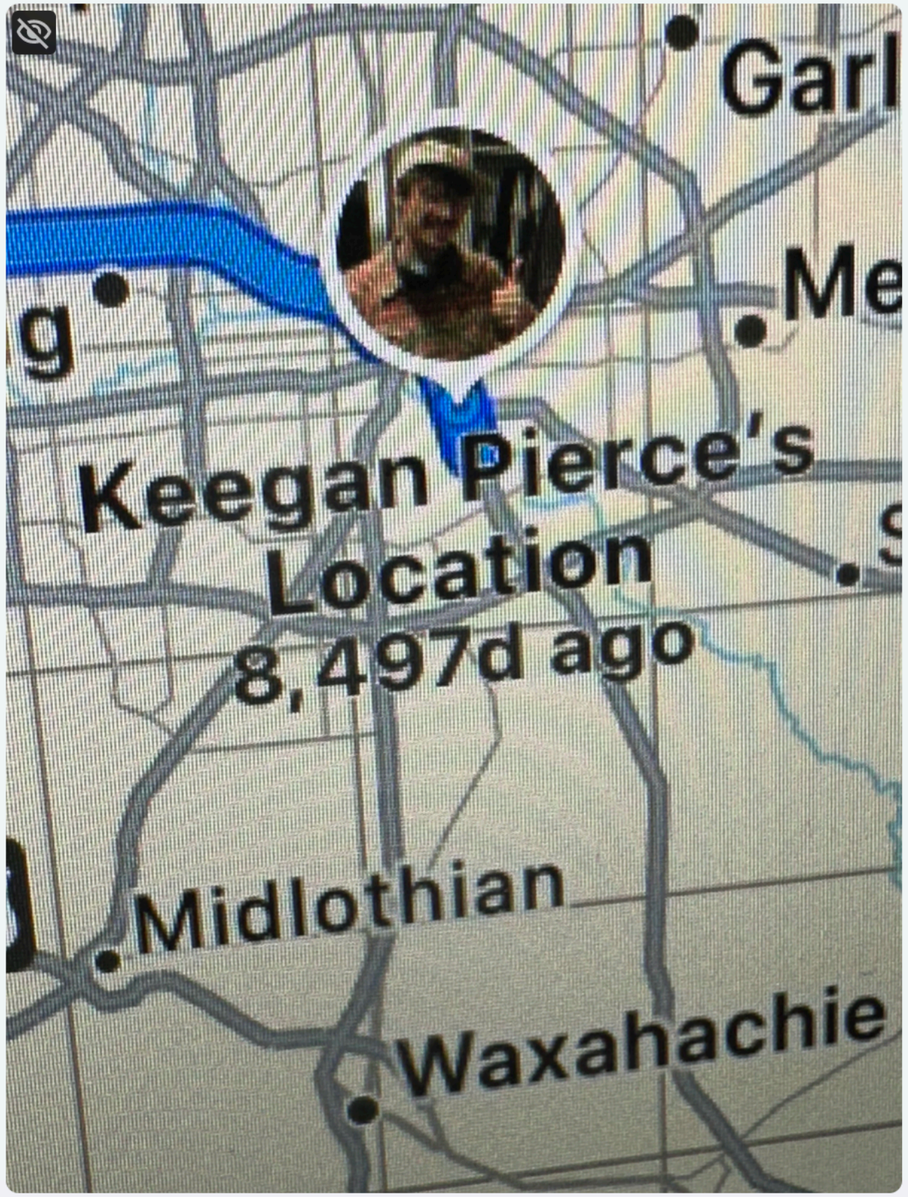
00:00:00 UTC ON 1 JANUARY 1970

`.timeIntervalSinceReferenceDate`

00:00:00 UTC ON 1 JANUARY 2001

Greg Pierce
@agiletortoise@mastodon.social

CarPlay found where my 22yo was the year before his birth.

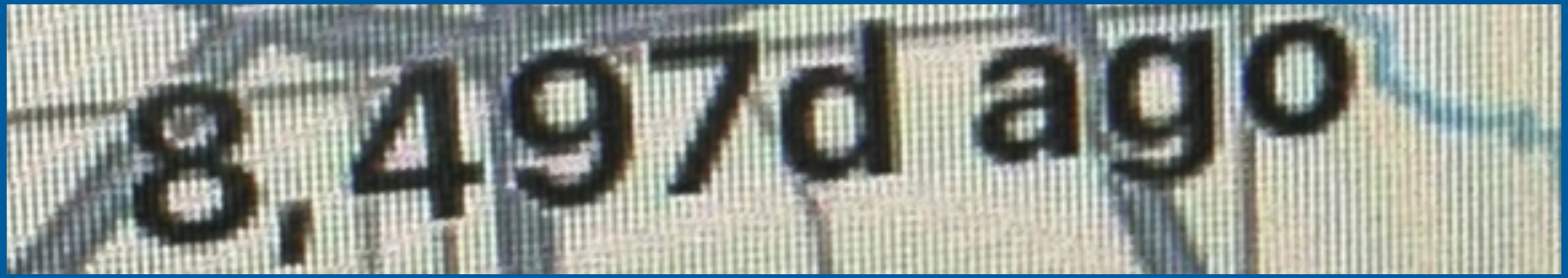


Apr 07, 2024, 06:07 PM · · Ivory for iOS

Apr 07, 2024, 06:07 PM

From and including: **Monday, 1 January 2001**
To, but **not** including **Sunday, 7 April 2024**

Result: 8497 days



Ellen Shapiro

@designatednerd

Apr 7

@agiletortoise .timeIntervalSinceReferenceDate strikes again

0

`.timeIntervalSince1970`

00:00:00 UTC ON 1 JANUARY 1970

`.timeIntervalSinceReferenceDate`

00:00:00 UTC ON 1 JANUARY 2001

INT32

Year 2038 problem

Article Talk

From Wikipedia, the free encyclopedia

The **year 2038 problem** (also known as **Y2038**,[1] **Y2K38**, **Y2K38 superbug** or the **Epochalypse**[2][3]) is a time computing problem that leaves some computer systems unable to represent times after 03:14:07 UTC on 19 January 2038.

https://en.wikipedia.org/wiki/Year_2038_problem

Year 2038 problem

Article Talk

From Wikipedia, the free encyclopedia

The **year 2038 problem** (also known as **Y2038**,[1] **Y2K38**, **Y2K38 superbug** or the **Epochalypse**[2][3]) is a time computing problem that leaves some computer systems unable to represent times after 03:14:07 UTC on 19 January 2038.

https://en.wikipedia.org/wiki/Year_2038_problem

.timeIntervalSince1970

00:00:00 UTC ON 1 JANUARY 1970

.timeIntervalSinceReferenceDate

00:00:00 UTC ON 1 JANUARY 2001

.timeIntervalBetween1970AndReferenceDate

.timeIntervalSince1970

00:00:00 UTC ON 1 JANUARY 1970

.timeIntervalSinceReferenceDate

00:00:00 UTC ON 1 JANUARY 2001

.timeIntervalBetween1970AndReferenceDate

978,307,200 SECONDS

```
previousSeconds += 1
```

CALENDAR

THE 🍎 DEFINITION

A DEFINITION OF THE RELATIONSHIPS BETWEEN CALENDAR UNITS AND ABSOLUTE POINTS IN TIME, PROVIDING FEATURES FOR CALCULATION AND COMPARISON OF DATES

MY DEFINITION

THE ONLY WAY TO RELATE AN ABSOLUTE POINT IN TIME TO ANYTHING HUMANS WILL ACTUALLY BE ABLE TO UNDERSTAND

404 DAY NOT FOUND

2024

All Calendars

January

SUN	MON	TUE	WED	THU	FRI	SAT
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

February

SUN	MON	TUE	WED	THU	FRI	SAT
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	1	2
3	4	5	6	7	8	9

March

SUN	MON	TUE	WED	THU	FRI	SAT
25	26	27	28	29	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

April

SUN	MON	TUE	WED	THU	FRI	SAT
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

May

SUN	MON	TUE	WED	THU	FRI	SAT
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

June

SUN	MON	TUE	WED	THU	FRI	SAT
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

July

SUN	MON	TUE	WED	THU	FRI	SAT
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

August

SUN	MON	TUE	WED	THU	FRI	SAT
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

September

SUN	MON	TUE	WED	THU	FRI	SAT
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5
6	7	8	9	10	11	12

October

SUN	MON	TUE	WED	THU	FRI	SAT
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

November

SUN	MON	TUE	WED	THU	FRI	SAT
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

December

SUN	MON	TUE	WED	THU	FRI	SAT
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

Calendar.Identifier.gregorian

PAINTED BY BARTOLOMEO PASSAROTTI

365.25

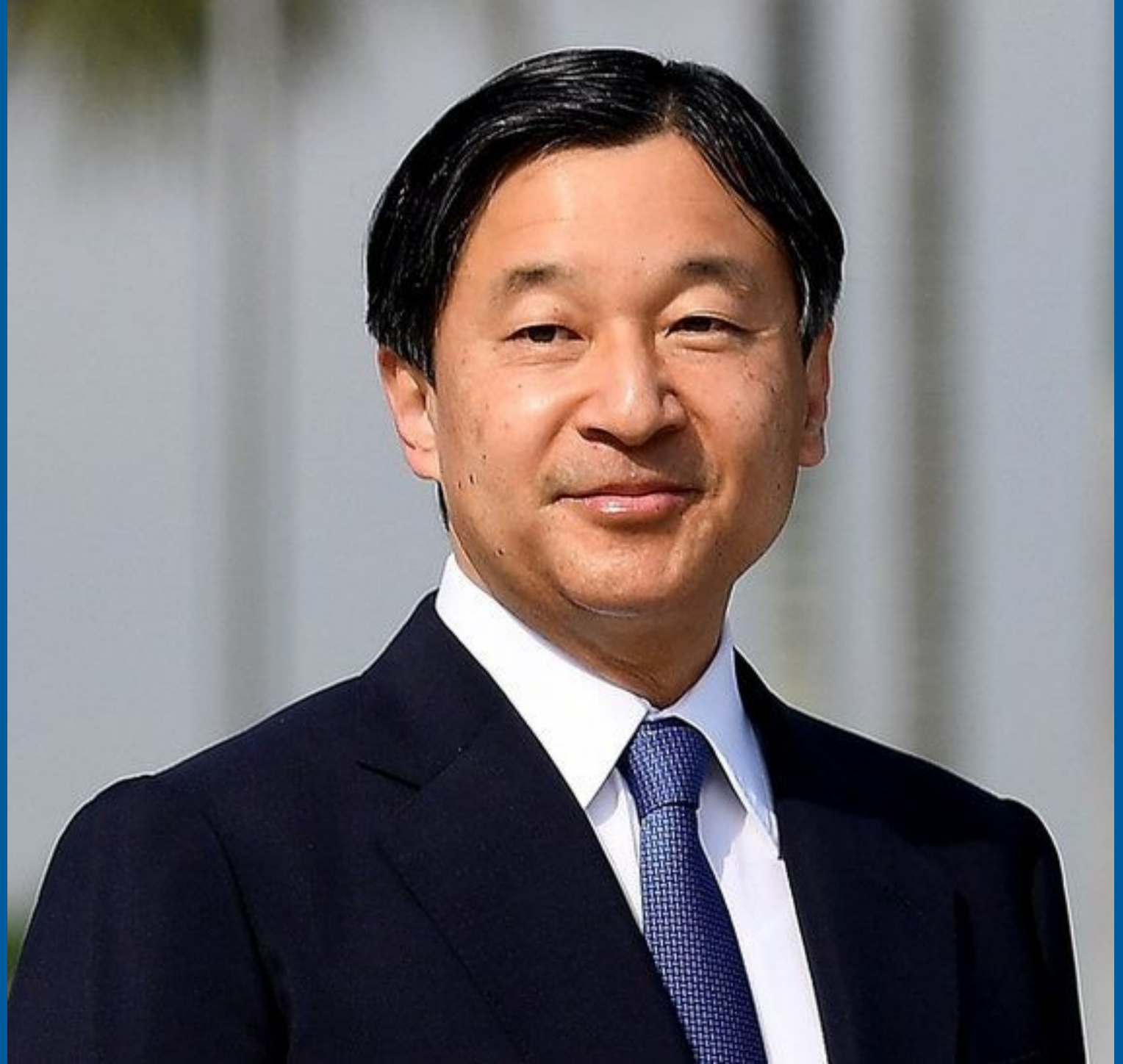
365.2425

PAINTED BY BARTOLOMEO PASSAROTTI

Code Blame 2107 lines (1783 loc) · 87.9 KB Raw

```
145     internal final class _CalendarGregorian: _CalendarProtocol, @unchecked Sendable {
157         init(identifier: Calendar.Identifier, timeZone: TimeZone?, locale: Locale?, firstWeekday:
    } else {
        self.gregorianStartYear = 1582
        self.julianCutoverDay = 2299161
        self.gregorianStartDate = Date(timeIntervalSince1970: -12219292800) // 1582-10-15T00:00:00Z
```

Calendar.Identifier.japan



Calendar.identifier.hebrew

Calendar.idenfier.islamic

Calendar.identifier.chinese

Calendar.identifier.indian

DO NOT TRY THIS (MANUALLY) AT HOME

```
Calendar(identifier: .gregorian)
```

Calendar.autoupdatingCurrent

DATE FORMATTER

THE 🍎 DEFINITION

INSTANCES OF DATEFORMATTER CREATE STRING REPRESENTATIONS OF DATE OBJECTS, AND CONVERT TEXTUAL REPRESENTATIONS OF DATES AND TIMES INTO DATE OBJECTS.

MY DEFINITION

THE THING YOU SHOULD USE TO TURN A DATE
INTO A USER-FACING STRING.

```
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent
```

```
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent
print(formatter.string(from: now))
```

```
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent
print(formatter.string(from: now))
// prints: (nothing!)
```

```swift
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent

formatter.dateFormat = "MM-dd-yyyy"
print(formatter.string(from: now))
// prints: "05-23-2024"
```

NSDateFormatter.com

Easy Skeezy Date Formatting for Swift and Objective-C

DATE INPUT:

01/19/2024, 12:30 AM

(enter in ISO8601 format)

FORMAT TEXT

EEEE, d MMM yyyy

LOCALE

en_US_POSIX

Result

Friday, 19 Jan 2024

```swift
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent

formatter.dateStyle = .full
print(formatter.string(from: now))
// prints: "Thursday, 23 May 2024"
```

```swift
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent

formatter.dateStyle = .full
print(formatter.string(from: now))
// prints: "Thursday, 23 May 2024"

formatter.locale = Locale(identifier: "en_US")
```

```swift
let now = Date()

let formatter = DateFormatter()
formatter.calendar = Calendar.autoupdatingCurrent

formatter.dateStyle = .full
print(formatter.string(from: now))
// prints: "Thursday, 23 May 2024"

formatter.locale = Locale(identifier: "en_US")
print(formatter.string(from: now))
// prints: "Thursday, May 23 2024"
```

```
let now = Date()

let aucFormatter = DateFormatter()
aucFormatter.calendar = Calendar.autoupdatingCurrent
print(formatter.string(from: now))
// prints: "Thursday, 23 May 2024"

let islamicFormatter = DateFormatter()
islamicFormatter.calendar = Calendar(identifier: .islamic)
```

```swift
let now = Date()

let aucFormatter = DateFormatter()
aucFormatter.calendar = Calendar.autoupdatingCurrent
print(formatter.string(from: now))
// prints: "Thursday, 23 May 2024"

let islamicFormatter = DateFormatter()
islamicFormatter.calendar = Calendar(identifier: .islamic)
islamicFormatter.dateStyle = .full
// prints "Thursday, 15 Dhu‘l-Qi‘dah, 1445 AH"
```

```swift
let now = Date()

let aucFormatter = DateFormatter()
aucFormatter.calendar = Calendar.autoupdatingCurrent
print(formatter.string(from: now))
// prints: "Friday, 19 January 2024"

let islamicFormatter = DateFormatter()
islamicFormatter.calendar = Calendar(identifier: .islamic)
islamicFormatter.dateStyle = .full
print(islamicFormatter.string(from: now))
// prints "Thursday, 15 Dhu‘l-Qi‘dah, 1445 AH"

islamicFormatter.locale = Locale(identifier: "en_US")
print(islamicFormatter.string(from: now))
// prints "Thursday, Dhu‘l-Qi‘dah 15, 1445 AH"
```

OTHER USEFUL DATE FORMATTERS

OTHER USEFUL DATE FORMATTERS

- `DateIntervalFormatter`

OTHER USEFUL DATE FORMATTERS

- `DateIntervalFormatter`
- `DateComponentsFormatter`

OTHER USEFUL DATE FORMATTERS

- `DateIntervalFormatter`
- `DateComponentsFormatter`
- `ISO8601DateFormatter`

OTHER USEFUL DATE FORMATTERS

- `DateIntervalFormatter`
- `DateComponentsFormatter`
- `ISO8601DateFormatter`
- `RelativeDateTimeFormatter` *

OTHER USEFUL DATE FORMATTERS

- `DateIntervalFormatter`
- `DateComponentsFormatter`
- `ISO8601DateFormatter`
- `RelativeDateTimeFormatter` *

* SWIFT ONLY

FORMATSTYLE

FORMATSTYLE

GOSHDARN

FORMATSTYLE.COM

```
Date(timeIntervalSinceReferenceDate: 0)
   .formatted()
// "1/1/2001, 12:00 AM"
```

VIA https://goshdarnformatstyle.com/

```swift
Date(timeIntervalSinceReferenceDate: 0)
   .formatted(.iso8601)
// "2001-01-01T00:00.000Z"
```

VIA https://goshdarnformatstyle.com/

```
let isoFormat = Date.ISO8601FormatStyle(
    dateSeparator: .dash,
    dateTimeSeparator: .standard,
    timeSeparator: .colon,
    timeZoneSeparator: .colon,
    includingFractionalSeconds: true,
    timeZone: TimeZone(identifier: "America/New York")
)
Date(timeIntervalSinceReferenceDate: 0)
  .formatted(isoFormat)
// "2000-12-31T19:00.000Z"
```

```swift
// TimeInterval
let referenceDay = Date(timeIntervalSinceReferenceDate: 0)
(referenceDay ..< referenceDay.addingTimeInterval(200))
  .formatted()
// "1/1/01, 12:00 – 12:03 AM"
```

VIA https://goshdarnformatstyle.com/

ALL THE TYPE INFERENCE

DATE COMPONENTS

THE 🍎 DEFINITION

A DATE OR TIME SPECIFIED IN TERMS OF UNITS (SUCH AS YEAR. MONTH. DAY. HOUR. AND MINUTE) TO BE EVALUATED IN A CALENDAR SYSTEM AND TIME ZONE.

MY DEFINITION

HOW TO DEFINE WHICH PARTS OF A DATE
YOU ACTUALLY CARE ABOUT

```
init(
    calendar: Calendar? = nil,
    timeZone: TimeZone? = nil,
    era: Int? = nil,
    year: Int? = nil,
    month: Int? = nil,
    day: Int? = nil,
    hour: Int? = nil,
    minute: Int? = nil,
    second: Int? = nil,
    nanosecond: Int? = nil,
    weekday: Int? = nil,
    weekdayOrdinal: Int? = nil,
    quarter: Int? = nil,
    weekOfMonth: Int? = nil,
    weekOfYear: Int? = nil,
    yearForWeekOfYear: Int? = nil
)
```


OPTIONALS!
LIGHTYEAR
OPTIONALS EVERYWHERE!
imgflip.com

```
init(
    calendar: Calendar? = nil,
    timeZone: TimeZone? = nil,
    era: Int? = nil,
    year: Int? = nil,
    month: Int? = nil,
    day: Int? = nil,
    hour: Int? = nil,
    minute: Int? = nil,
    second: Int? = nil,
    nanosecond: Int? = nil,
    weekday: Int? = nil,
    weekdayOrdinal: Int? = nil,
    quarter: Int? = nil,
    weekOfMonth: Int? = nil,
    weekOfYear: Int? = nil,
    yearForWeekOfYear: Int? = nil
)
```

```
dateComponents(_ components: Set<Calendar.Component>,
               from date: Date) -> DateComponents

date(from: DateComponents) -> Date
```

GETTING COMPONENTS OUT OF A DATE

```
let talkTime = ISO8601DateFormatter()
    .date(from: "2024-05-23T14:30:00+01:00")!
```

```
let talkTime = ISO8601DateFormatter()
    .date(from: "2024-05-23T14:30:00+01:00")!
let timeComponents = gregorianCalendar
    .dateComponents([.hour, .minute], from: talkTime)
```

```
let talkTime = ISO8601DateFormatter()
    .date(from: "2024-05-23T14:30:00+01:00")!
let timeComponents = gregorianCalendar
    .dateComponents([.hour, .minute], from: talkTime)

let hour = timeComponents.hour
print(hour) // prints "Optional(14)"
let minute = timeComponents.minute
print(minute) // prints "Optional(30)"
```

```
let talkTime = ISO8601DateFormatter()
    .date(from: "2024-05-23T14:30:00+01:00")!
let timeComponents = gregorianCalendar
    .dateComponents([.hour, .minute], from: talkTime)

let hour = timeComponents.hour
print(hour) // prints "Optional(14)"
let minute = timeComponents.minute
print(minute) // prints "Optional(30)"

let second = timeComponents.second
```

```
let talkTime = ISO8601DateFormatter()
    .date(from: "2024-05-23T14:30:00+01:00")!
let timeComponents = gregorianCalendar
    .dateComponents([.hour, .minute], from: talkTime)

let hour = timeComponents.hour
print(hour) // prints "Optional(14)"
let minute = timeComponents.minute
print(minute) // prints "Optional(10)"

let second = timeComponents.second
print(second) // prints "nil"
```


You had to ask.

TURNING COMPONENTS INTO A DATE

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)
```

```
let components = DateComponents(year: 2024,
                                month: 4,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
gregorianFormatter.timeStyle = .short
print(gregorianFormatter.string(from: gregorianResult))
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
gregorianFormatter.timeStyle = .short
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024 at 12:00AM"
```

```swift
let components = DateComponents(year: 2024)

let gregorianResult = gregorianCalendar.date(from: components)!
gregorianFormatter.timeStyle = .short
print(gregorianFormatter.string(from: gregorianResult))
// prints "Monday, 1 January 2024 at 12:00 AM"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23,
                                hour: 14,
                                minute: 30)

let gregorianResult = gregorianCalendar.date(from: components)!
gregorianFormatter.timeStyle = .short
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024 at 2:30 PM"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
print(gregorianFormatter.string(from: islamicResult))
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thusrday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
print(gregorianFormatter.string(from: islamicResult))
// prints "Thursday, 8 September 2585"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
print(gregorianFormatter.string(from: islamicResult))
// prints "Thursday, 8 September 2585"

print(islamicFormatter.string(from: islamicResult))
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
print(gregorianFormatter.string(from: islamicResult))
// prints "Thursday, 8 September 2585"

print(islamicFormatter.string(from: islamicResult))
// prints "Thursday, Jumada I 23, 2024 AH"
```

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024"

let islamicResult = islamicCalendar.date(from: components)!
print(gregorianFormatter.string(from: islamicResult))
// prints "Thursday, 8 September 2585"

print(islamicFormatter.string(from: islamicResult))
// prints "Thursday, Jumada I 23, 2024 AH" ⬅️
```

```
var calendar: Calendar?
```

The calendar used to interpret the other values in this structure.


¯_(ツ)_/¯

```
let components = DateComponents(year: 2024,
                                month: 5,
                                day: 23)

let gregorianResult = gregorianCalendar.date(from: components)!
print(gregorianFormatter.string(from: gregorianResult))
// prints "Thursday, 23 May 2024

print(islamicFormatter.string(from: gregorianResult))
// prints "Thursday, Dhu‘l-Qi‘dah 15, 1445 AH"
```

.timeIntervalSince1970

00:00:00 UTC ON 1 JANUARY 1970

.timeIntervalSinceReferenceDate

00:00:00 UTC ON 1 JANUARY 2001

.timeIntervalBetween1970AndReferenceDate

978,307,200 SECONDS

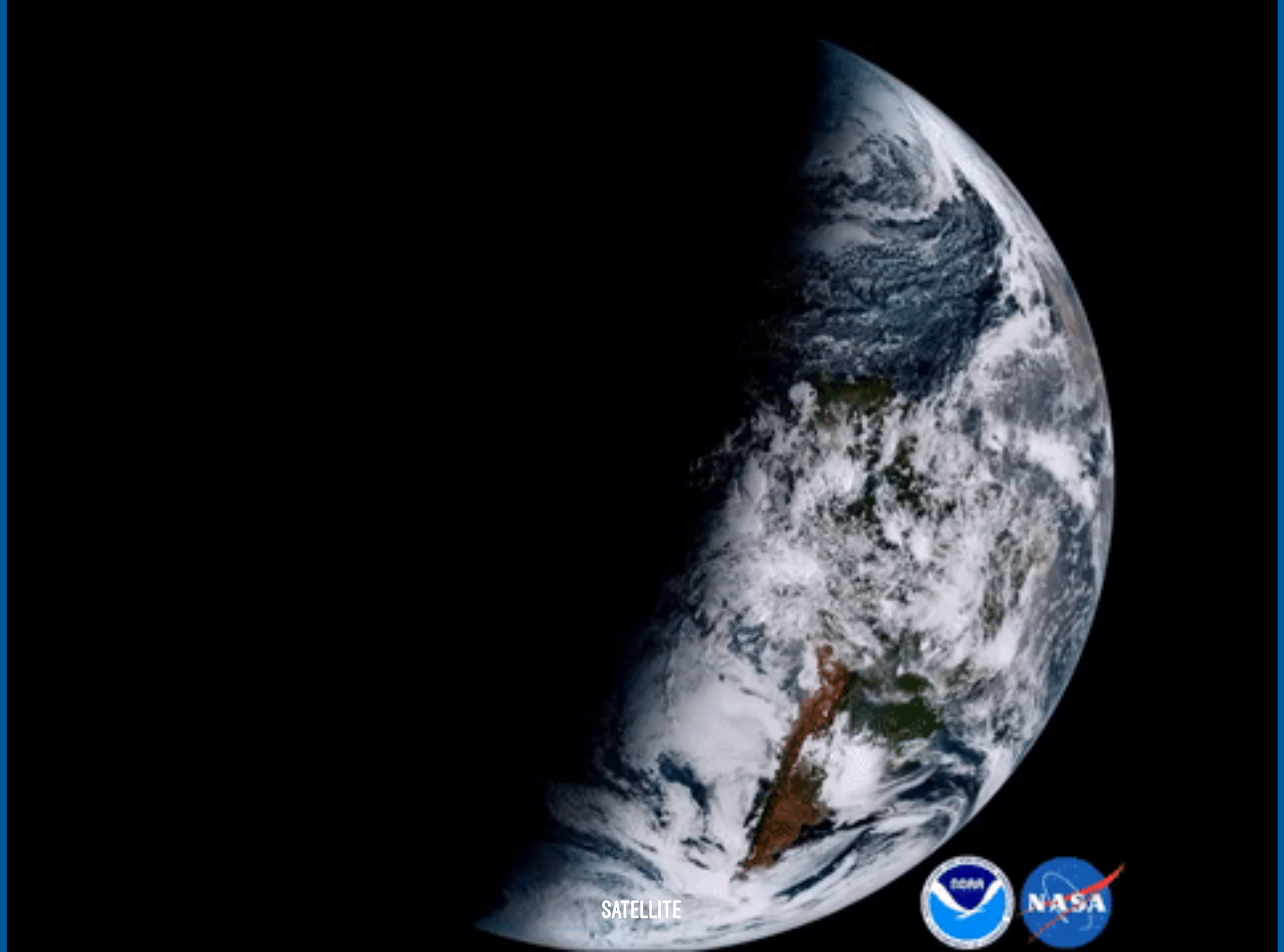
UTC

UTC

COORDINATED UNIVERSAL TIME

UTC

UNIVERSAL TIME. COORDINATED!



GOES-16

TIME ZONE

THE 🍎 DEFINITION

INFORMATION ABOUT STANDARD TIME CONVENTIONS ASSOCIATED WITH A SPECIFIC GEOPOLITICAL REGION.

MY DEFINITION

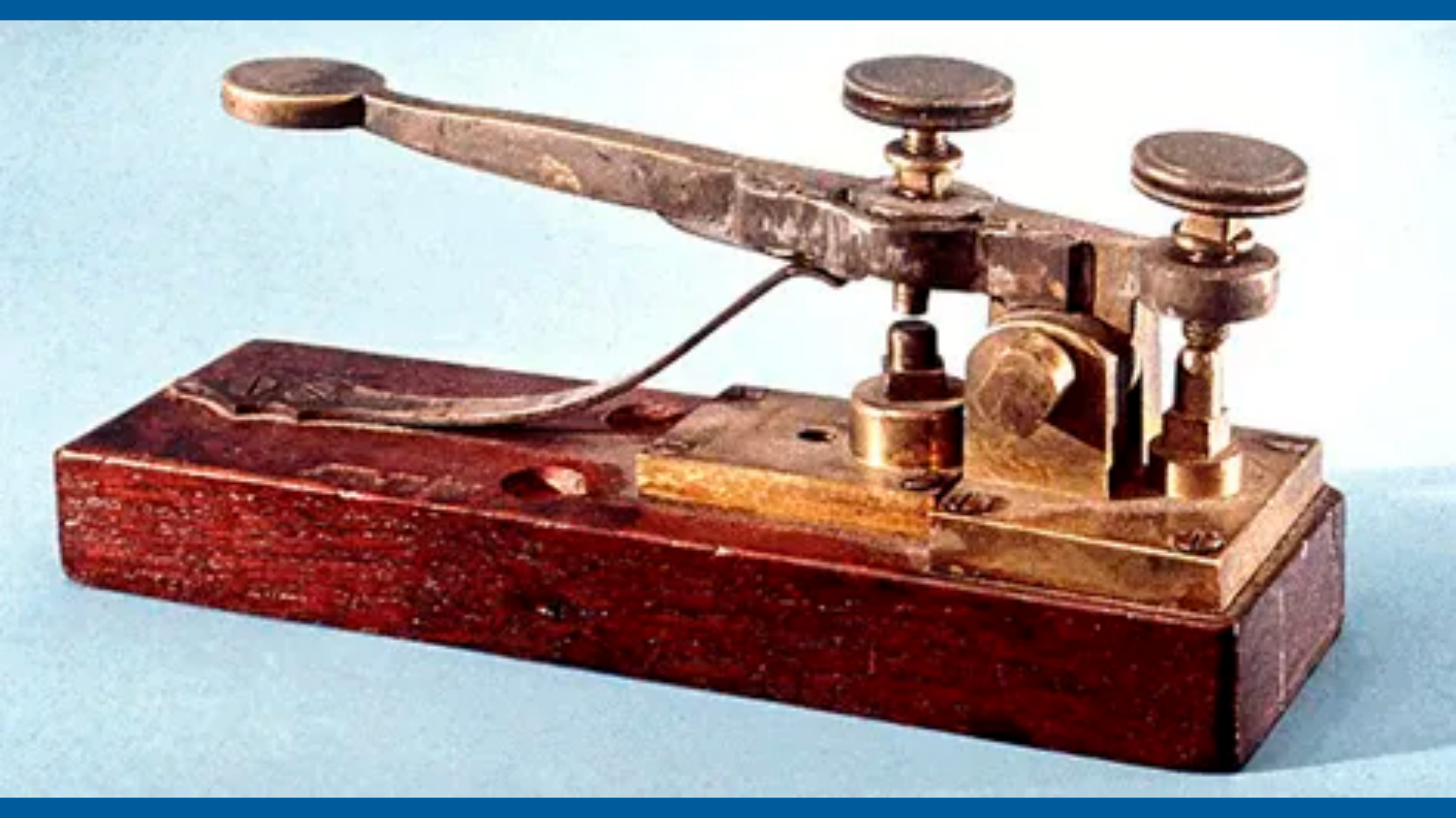
THE WAY HUMANS RECONCILE THE ROTATION OF THE EARTH AND A NEED TO COORDINATE TIME ACROSS PHYSICAL AND POLITICAL BOUNDARIES

BOSTON: 71.0552° W

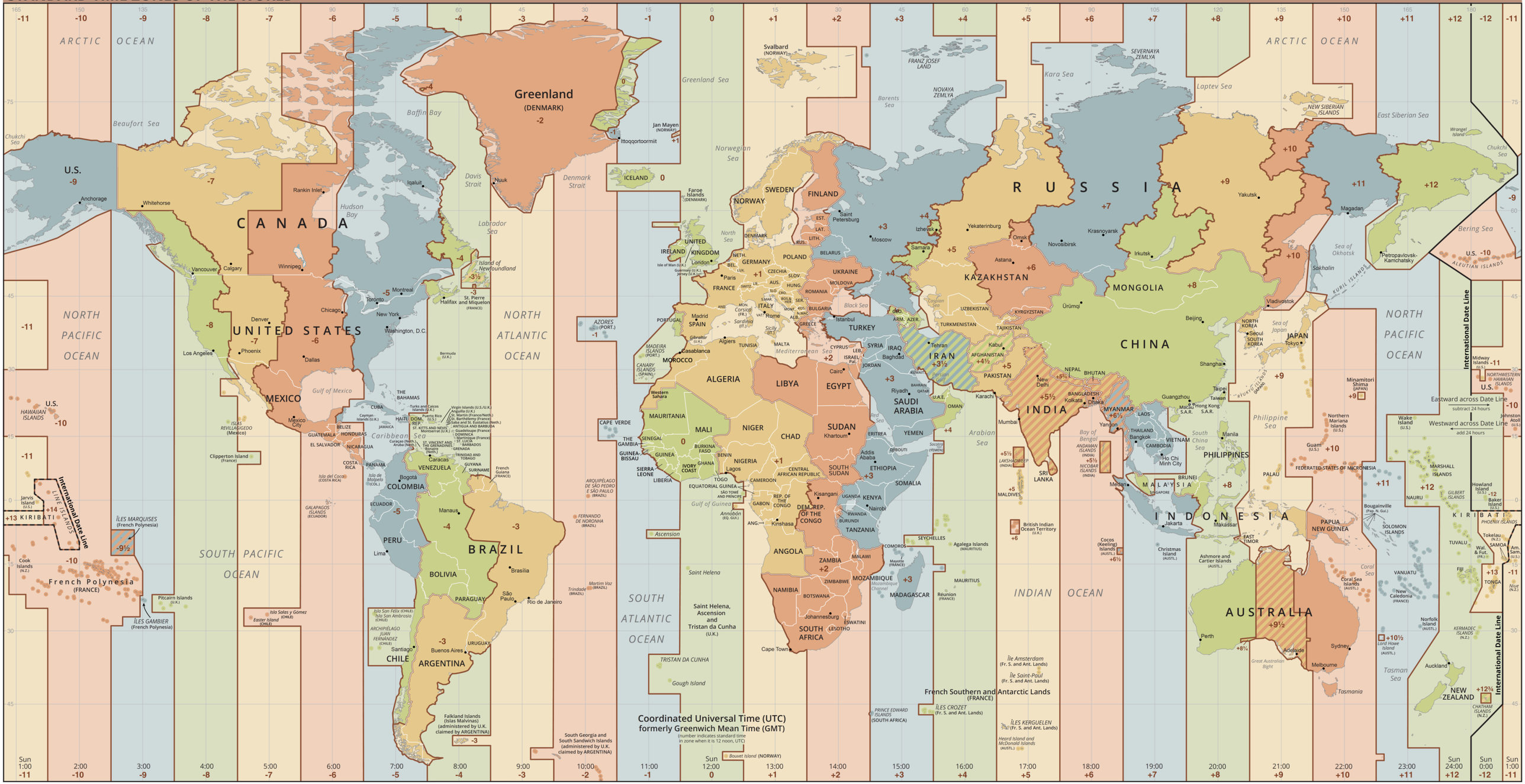
NYC: 73.9935° W



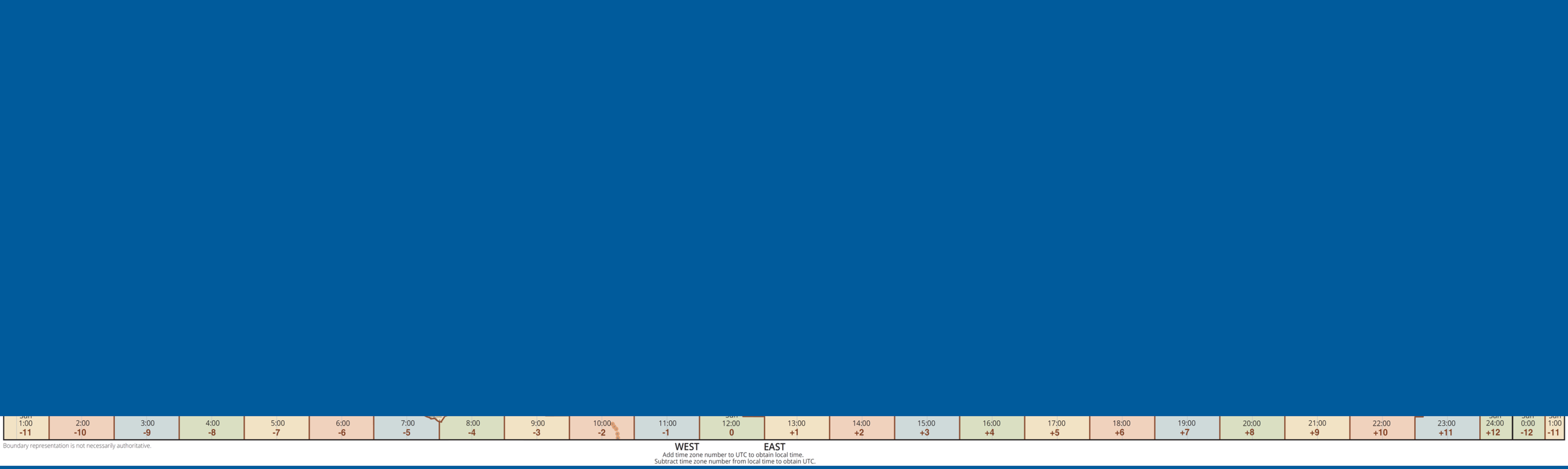

GMT



STANDARD TIME ZONES OF THE WORLD



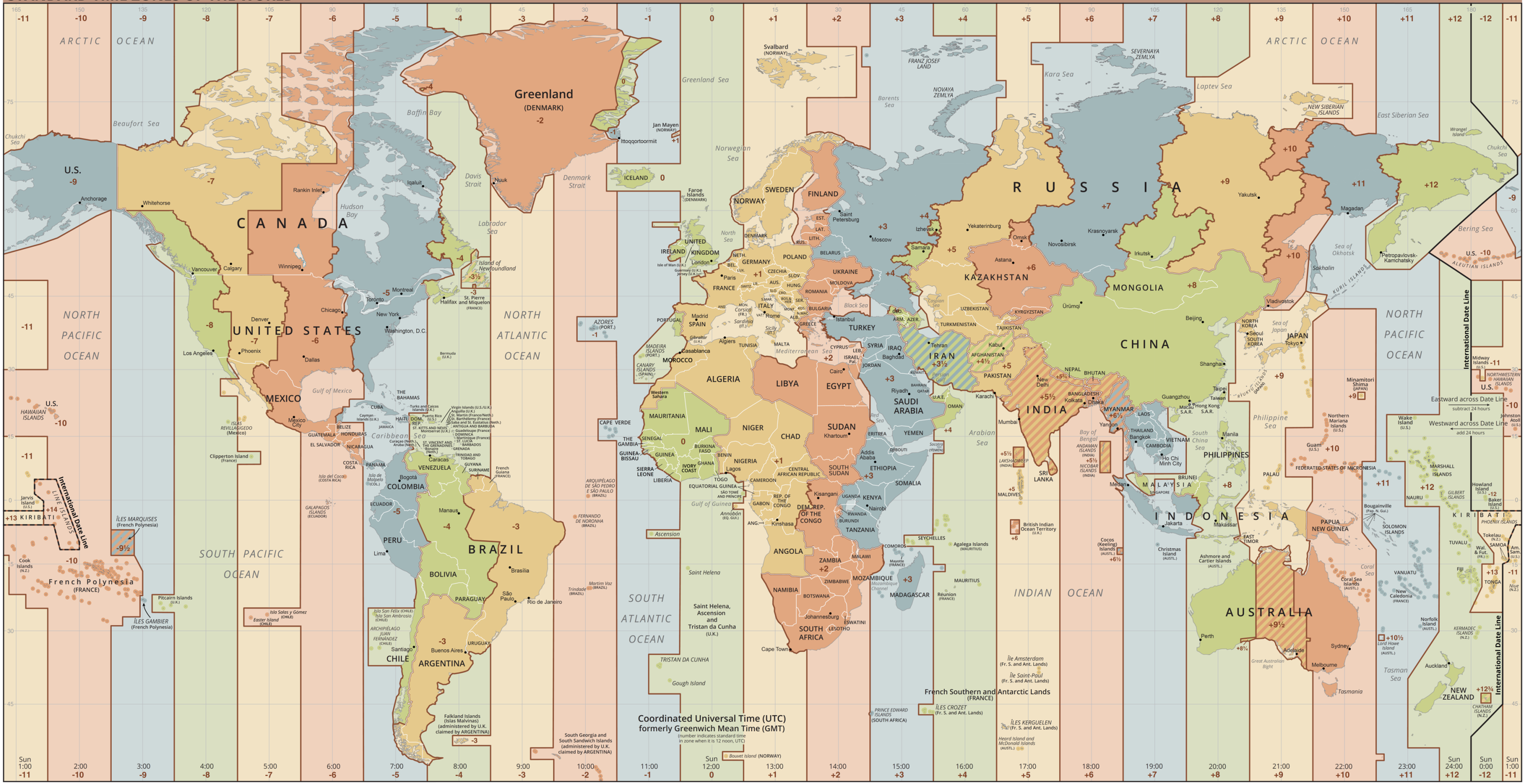
Boundary representation is not necessarily authoritative.



Boundary representation is not necessarily authoritative.

$$360 / 24 = 15$$

STANDARD TIME ZONES OF THE WORLD



Boundary representation is not necessarily authoritative.


B A T I
PHOENIX ISLANDS
Tokelau
(N.Z.)
SAMOA
Wal.
& Fut.
(FR.)
Am.
Sam.
(U.S.)
15
+13
-11
TONGA
Niue
(N.Z.)
30
International Date Line
+12¾

Samoa and Tokelau skip a day for dateline change

30 December 2011 · Comments

https://www.bbc.com/news/world-asia-16351377

Samoa and Tokelau skip a day for dateline change

30 December 2011 · Comments

MY DEFINITION

THE WAY HUMANS RECONCILE THE ROTATION OF THE EARTH AND A NEED TO COORDINATE TIME ACROSS PHYSICAL AND POLITICAL BOUNDARIES

MY DEFINITION

THE WAY HUMANS RECONCILE THE ROTATION OF THE EARTH*
AND A NEED TO COORDINATE TIME ACROSS PHYSICAL
AND POLITICAL BOUNDARIES

* - MOSTLY

LTC

COORDINATED LUNAR TIME

APPLICATIONS

Telling time on the Moon

A new era of lunar exploration is on the rise, with dozens of Moon missions planned for the coming decade. Europe is in the forefront here, contributing to building the Gateway lunar station and the Orion spacecraft – set to return humans to our natural satellite – as well as developing its large logistic lunar lander, known as Argonaut. As dozens of missions will be operating on and around the Moon and needing to communicate together and fix their positions independently from Earth, this new era will require its own time.

https://www.esa.int/Applications/Satellite_navigation/Telling_time_on_the_Moon

EXECUTIVE OFFICE OF THE PRESIDENT
OFFICE OF SCIENCE AND TECHNOLOGY POLICY
WASHINGTON, D.C. 20502

April 2, 2024

MEMORANDUM FOR DEPARTMENTS AND AGENCIES PARTICIPATING IN THE WHITE HOUSE CISLUNAR TECHNOLGY STRATEGY INTERAGENCY WORKING GROUP

FROM: Arati Prabhakar, Assistant to the President for Science and Technology and Director, Office of Science and Technology Policy

SUBJECT: Policy on Celestial Time Standardization in Support of the National Cislunar Science and Technology (S&T) Strategy

This memorandum outlines the Biden-Harris Administration's policy to establish time standards at and around celestial bodies other than Earth to advance the National Cislunar S&T Strategy.[1] OSTP directs federal departments and agencies to align their planning and policies with this memorandum.

SERIOUSLY.
DO NOT TRY THIS
(MANUALLY) AT HOME

```
let allTimeZones = TimeZone.knownTimeZoneIdentifiers
    .compactMap { TimeZone(identier: $0) }
```

https://commons.wikimedia.org/wiki/File:China_%E2%80%93_U.S._area_comparison.jpg

America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac

America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia

TimeZone.autoupdatingCurrent

```swift
let utcTimeZone = TimeZone(identifier: "UTC")!
```

Depart New York Jan 14 Terminal 4

10:20 PM EST

Duration 18 hours, 35 minutes

Arrive Singapore Jan 16

5:55 AM GMT+8

Depart New York Jan 14 Terminal 4

10:20 PM EST

Duration 18 hours, 35 minutes

Arrive Singapore Jan 16

5:55 AM GMT+8

Depart New York Jan 14 Terminal 4

10:20 PM EST

Duration 18 hours, 35 minutes

Arrive Singapore Jan 16

5:55 AM GMT+8

Depart New York Jan 14 Terminal 4

10:20 PM EST

Duration 18 hours, 35 minutes

Arrive Singapore Jan 16

5:55 AM GMT+8

View and/or edit details in TripIt : https://www.tripit.com/trip/show/id/338647098

10:20 PM EST
[Flight] JFK to SIN

Singapore Airlines 23, Terminal 4

Crosses the International Date Line

Tue, Jan 16
5:55 AM +08
Arrive Singapore (SIN)

View and/or edit details in TripIt : https://www.tripit.com/trip/show/id/338647098

10:20 PM EST
[Flight] JFK to SIN

Singapore Airlines 23, Terminal 4

Crosses the International Date Line

Tue, Jan 16
5:55 AM +08
Arrive Singapore (SIN)

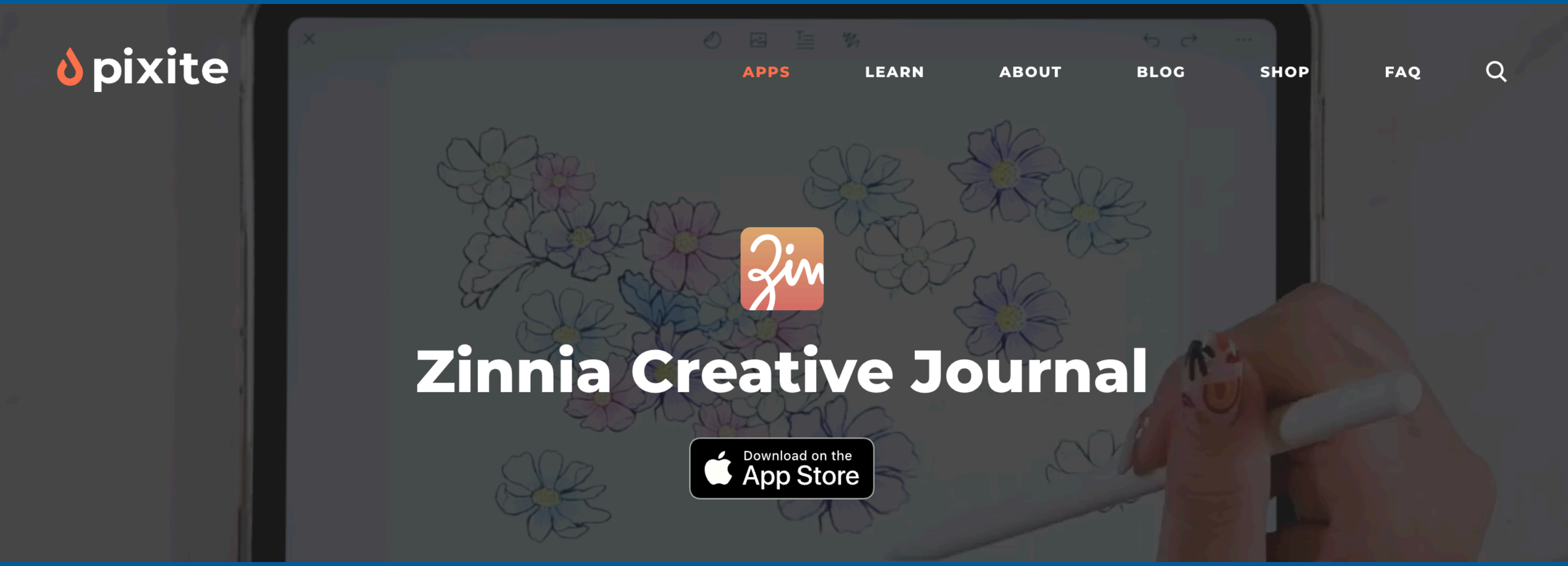
```
let utcTimeZone = TimeZone(identifier: "UTC")!
print(utcTimeZone.identifier)
```

```
let utcTimeZone = TimeZone(identifier: "UTC")!
print(utcTimeZone.identifier)
// prints: "GMT"
```

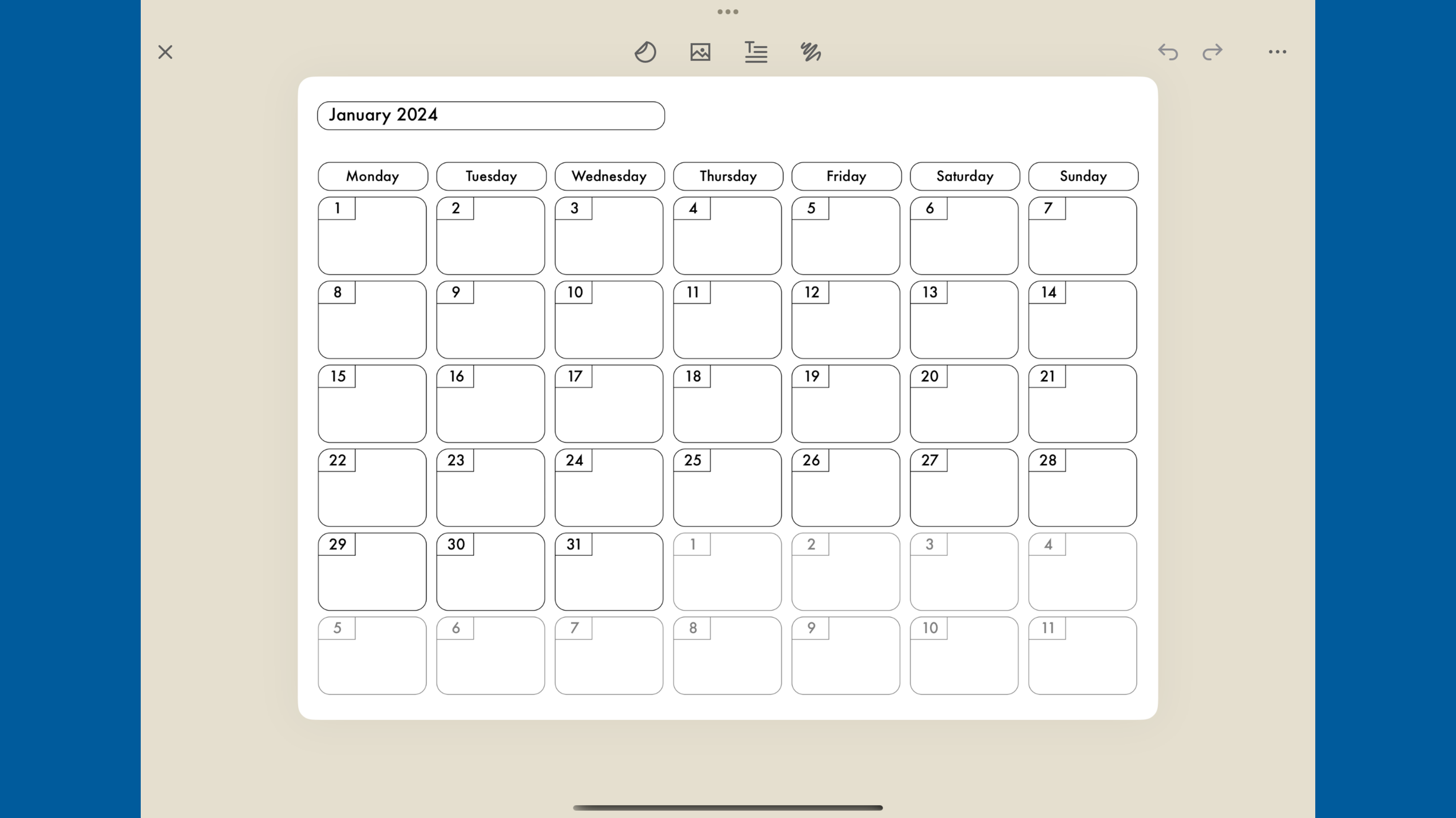
```
TimeZone.secondsFromGMT(for: Date)
```

GENERAL ADVICE

STORE ONLY THE COMPONENTS YOU NEED



HTTPS://WWW.PIXITEAPPS.COM/APPS/PLANNER-JOURNAL-ZINNIA


January 2024
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
1
2
3
4
5
6
7
8
9
10
11

THIS IS VERY EASY
TO OVERTHINK

THIS IS ALSO VERY EASY
TO UNDERESTIMATE

TEST YOUR DATES AND TIMES

CREATING A DATE IN ONE TIME ZONE READING IT IN ANOTHER

WATCH OUT FOR WHAT TIME
YOUR CI THINKS IT IS

ONE COMPONENT IS SUDDENLY LOWER

ONE COMPONENT IS SUDDENLY LOWER

- DAY ROLLOVERS

ONE COMPONENT IS SUDDENLY LOWER

- DAY ROLLOVERS
- MONTH ROLLOVERS

ONE COMPONENT IS SUDDENLY LOWER

- DAY ROLLOVERS
- MONTH ROLLOVERS
- YEAR ROLLOVERS

OCCASIONAL WEIRDNESS

OCCASIONAL WEIRDNESS

- LEAP YEAR ROLLOVERS

OCCASIONAL WEIRDNESS

- LEAP YEAR ROLLOVERS
- YEARS WITH EXTRA MONTHS IN LUNISOLAR CALENDARS

OCCASIONAL WEIRDNESS

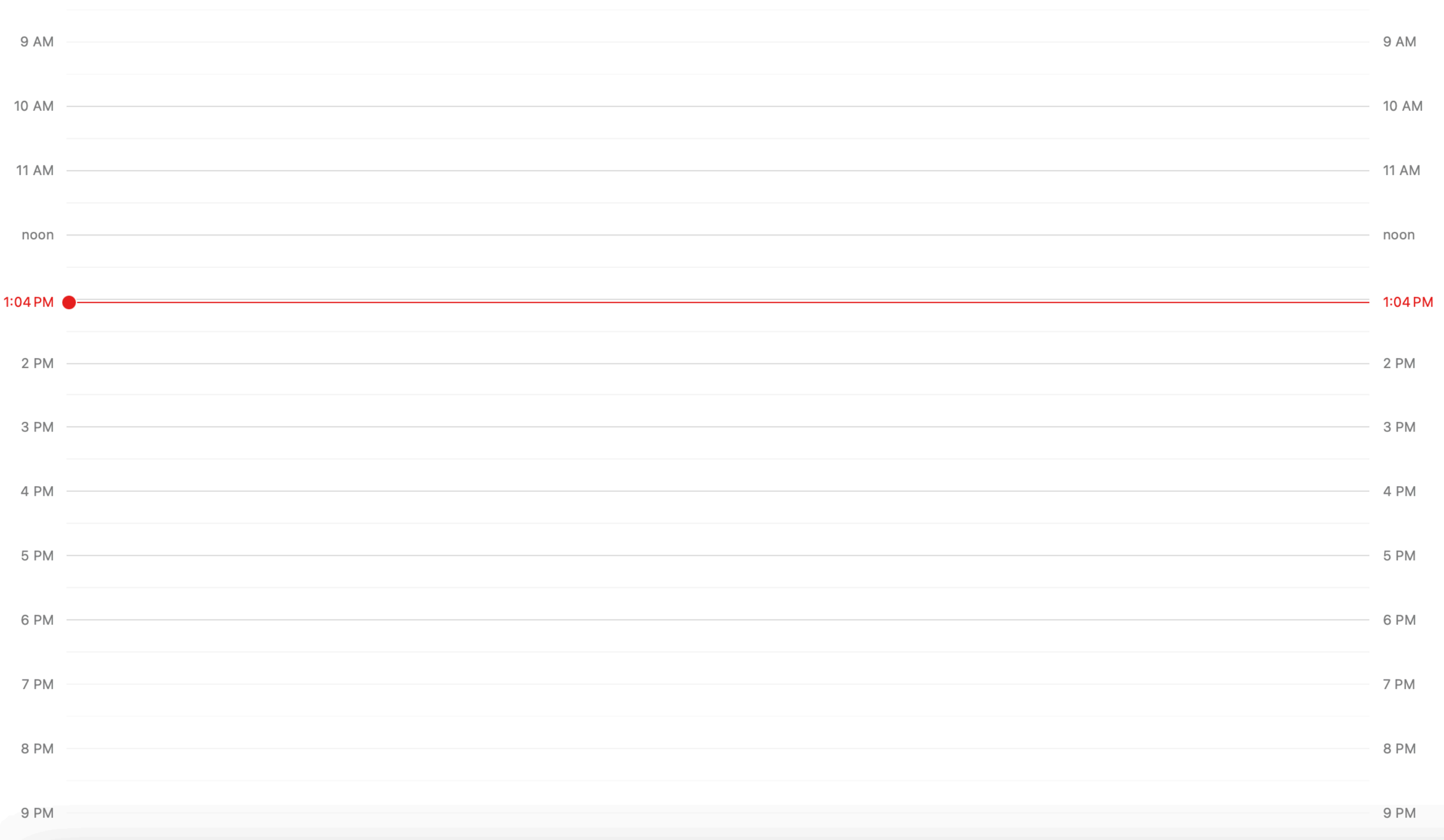
- LEAP YEAR ROLLOVERS
- YEARS WITH EXTRA MONTHS IN LUNISOLAR CALENDARS
- DAYLIGHT SAVINGS TIME ROLLOVERS

THINGS I SCREWED UP EVEN

AS I WROTE THIS TALK

CALENDAR + DATECOMPONENTS

```
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```


9 AM
10 AM
11 AM
noon
1:04 PM
2 PM
3 PM
4 PM
5 PM
6 PM
7 PM
8 PM
9 PM
9 AM
10 AM
11 AM
noon
1:04 PM
2 PM
3 PM
4 PM
5 PM
6 PM
7 PM
8 PM
9 PM

```swift
extension Date {
    func formattedNextHoursAdjsutingComponents(count: Int) -> [String] {
        var components = Calendar.autoupdatingCurrent
                                 .dateComponents([.year, .month,
                                                  .day, .hour,
                                                  .minute],
                                                 from: self)
        let originalHour = components.hour!
        var formattedHours = [String]()
        for hour in 0..<count {
            components.hour = originalHour + hour
            let updatedDate = Calendar.autoupdatingCurrent
                                      .date(from: components)
            let formatted = formatter.string(from: updatedDate)
            formattedHours.append(formatted)
        }

        return formattedHours
    }
}
```

```swift
extension Date {
    func formattedNextHoursAdjsutingComponents(count: Int) -> [String] {
        var components = Calendar.autoupdatingCurrent
                                 .dateComponents([.year, .month,
                                                  .day, .hour,
                                                  .minute],
                                                 from: self)
        let originalHour = components.hour!
        var formattedHours = [String]()
        for hour in 0..<count {
            components.hour = originalHour + hour
            let updatedDate = Calendar.autoupdatingCurrent
                                      .date(from: components)
            let formatted = formatter.string(from: updatedDate)
            formattedHours.append(formatted)
        }

        return formattedHours
    }
}
```

```swift
func formattedNextHoursAddingComponents(count: Int) -> [String] {
    var formattedHours = [String]()
    for hour in 0..<count {
        let updatedDate = Calendar.autoupdatingCurrent
            .date(byAdding: DateComponents(hour: hour), to: self)!

        let formatted = formatter.string(from: updatedDate)
        formattedHours.append(formatted)
    }

    return formattedHours
}
```

```
func formattedNextHoursAddingComponents(count: Int) -> [String] {
    var formattedHours = [String]()
    for hour in 0..<count {
        let updatedDate = Calendar.autoupdatingCurrent
            .date(byAdding: DateComponents(hour: hour), to: self)!

        let formatted = formatter.string(from: updatedDate)
        formattedHours.append(formatted)
    }

    return formattedHours
}
```

```swift
let now = Date()

let nowAdjustingComponents = now
    .formattedNextHoursAdjustingComponents(count: 12)
    .joined(separator: "\n")

let nowAddingComponents = now
    .formattedNextHoursAddingComponents(count: 12)
    .joined(separator: "\n")
```

```swift
// March 9, 2024 at 7pm Eastern Time
let springForward = ISO8601DateFormatter().date(from: "2024-03-10T00:00:00Z")!
formatter.timeZone = TimeZone(identifier: "America/New_York")!

let springForwardAdjustingComponents = springForward
    .formattedNextHoursAdjustingComponents(count: 12)
    .joined(separator: "\n")

let springForwardAddingComponents = springForward
    .formattedNextHoursAddingComponents(count: 12)
    .joined(separator: "\n")
```

NOW

ADJUSTING

Now by adjusting components:

1:16 PM
2:16 PM
3:16 PM
4:16 PM
5:16 PM
6:16 PM
7:16 PM
8:16 PM
9:16 PM
10:16 PM
11:16 PM
12:16 AM

ADDING

Now by adding components:

1:16 PM
2:16 PM
3:16 PM
4:16 PM
5:16 PM
6:16 PM
7:16 PM
8:16 PM
9:16 PM
10:16 PM
11:16 PM
12:16 AM

ADJUSTING

Now by adjusting components:

1:16 PM
2:16 PM
3:16 PM
4:16 PM
5:16 PM
6:16 PM
7:16 PM
8:16 PM
9:16 PM
10:16 PM
11:16 PM
12:16 AM

ADDING

Now by adding components:

1:16 PM
2:16 PM
3:16 PM
4:16 PM
5:16 PM
6:16 PM
7:16 PM
8:16 PM
9:16 PM
10:16 PM
11:16 PM
12:16 AM

SPRING FORWARD

ADJUSTING

Spring forward by adjusting components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM

ADDING

Spring forward by adding components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM

ADJUSTING

Spring forward by adjusting components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM

ADDING

Spring forward by adding components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM

ADJUSTING

```
Spring forward by adjusting components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
```

ADDING

```
Spring forward by adding components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM
```

ADJUSTING

```
Spring forward by adjusting components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
```

ADDING

```
Spring forward by adding components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM
```

ADJUSTING

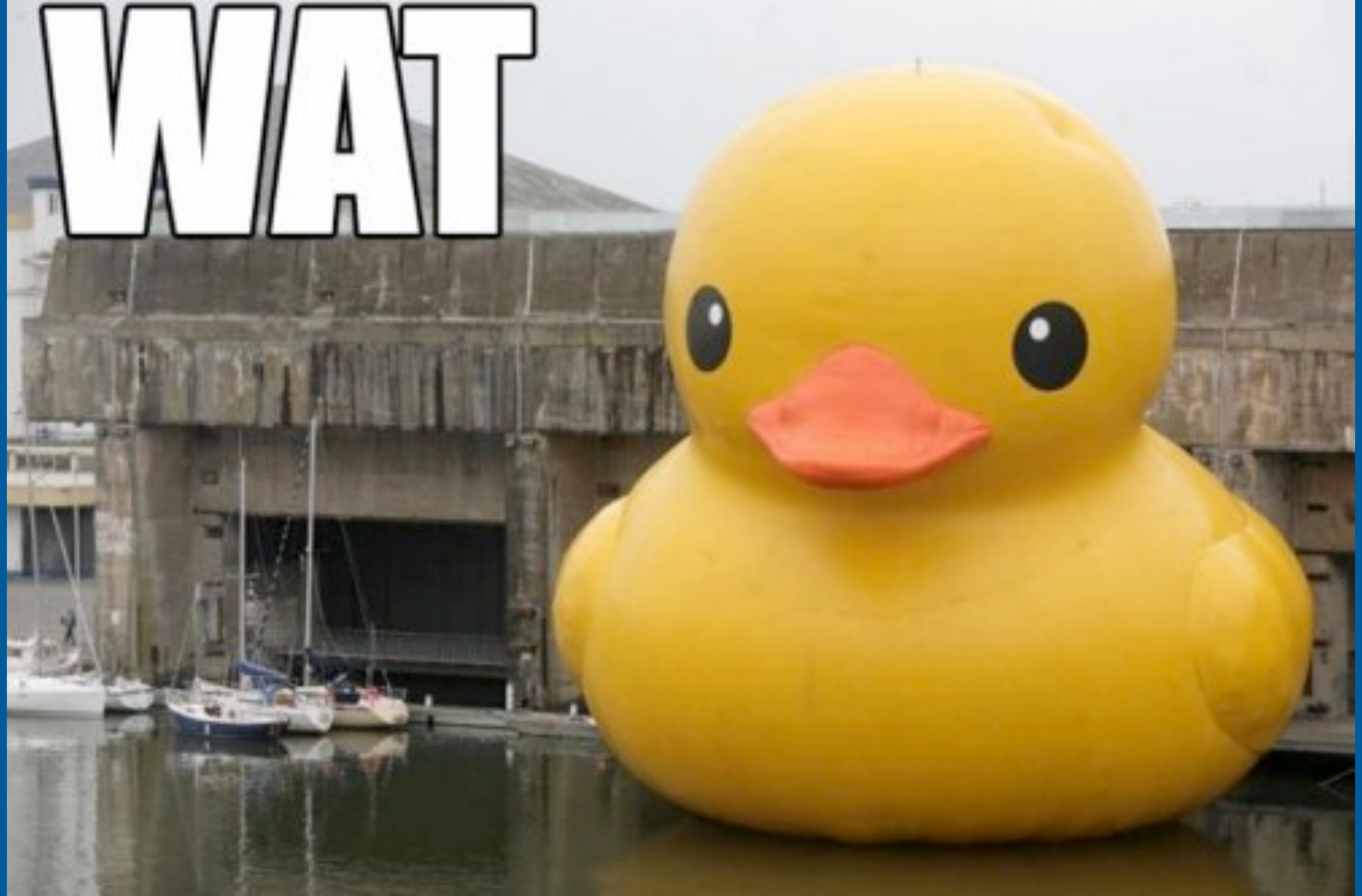
Spring forward by adjusting components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM

ADDING

Spring forward by adding components:

7:00 PM
8:00 PM
9:00 PM
10:00 PM
11:00 PM
12:00 AM
1:00 AM
3:00 AM
4:00 AM
5:00 AM
6:00 AM
7:00 AM


WAT


Dave DeLong – Senio
@scale
emerge
Lickability
RocketSurf
BLUETHRONE
@scale
USSP
Dave DeLong : The Temporal Axis of Space-Time

Your Calendrical Fallacy Is...

Helping you navigate the insane complexity of calendrically correct date and time operations

CALENDAR + DATECOMPONENTS

```
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```

CALENDAR + DATECOMPONENTS

```
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```

CALENDAR + DATECOMPONENTS

```
/// Find the closest matching date to the given date components
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```

March 9 2024 25:00

March 9 2024 25:00 ->
    March 10 2024 01:00

```
March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00
```

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00

```
March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
```

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match?

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match? ->
    March 10 2024 03:00

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match? ->
    March 10 2024 03:00

March 9 2024 27:00

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match? ->
    March 10 2024 03:00

March 9 2024 27:00 ->
    March 10 2024 03:00

```
March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match? ->
    March 10 2024 03:00

March 9 2024 27:00 ->
    March 10 2024 03:00 ->
    March 10 2024 03:00
```

March 9 2024 25:00 ->
    March 10 2024 01:00 ->
    March 10 2024 01:00

March 9 2024 26:00 ->
    March 10 2024 02:00 ->
    Does not exist, what's the closest match? ->
    March 10 2024 03:00 👯

March 9 2024 27:00 ->
    March 10 2024 03:00 ->
    March 10 2024 03:00 👯

CALENDAR + DATECOMPONENTS

```
/// Find the closest matching date to the given date components
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```

CALENDAR + DATECOMPONENTS

```
/// Find the closest matching date to the given date components
Calendar.autoupdatingCurrent.date(from dateComponents: DateComponents)

/// Take the passed-in date and add the passed-in components
Calendar.autoupdatingCurrent.date(byAdding dateComponents: DateComponents,
                                  to date: Date)
```

March 9 2024 17:00 + 8h

March 9 2024 17:00 + 8h ->
March 10 2024 01:00

March 9 2024 17:00 + 8h ->
    March 10 2024 01:00

March 9 2024 17:00 + 9h

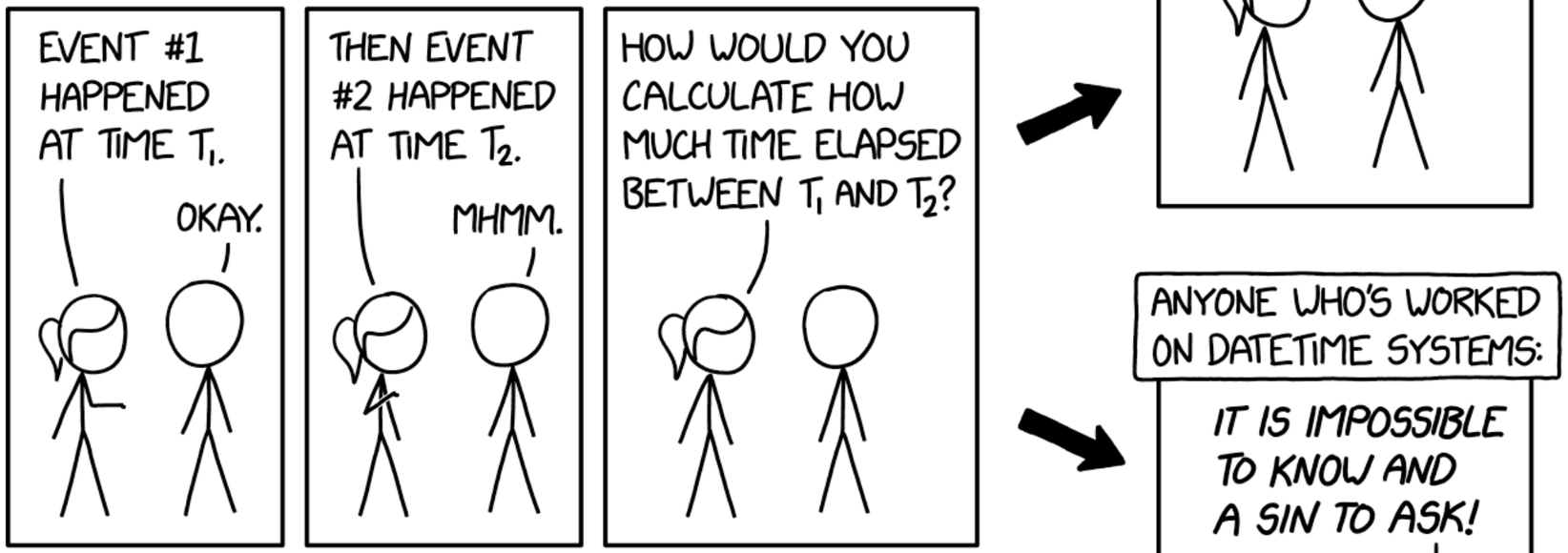
```
March 9 2024 17:00 + 8h ->
     March 10 2024 01:00

March 9 2024 17:00 + 9h ->
     March 10 2024 03:00
```

```
March 9 2024 17:00 + 8h ->
     March 10 2024 01:00

March 9 2024 17:00 + 9h ->
     March 10 2024 03:00

March 9 2024 17:00 + 10h
```

March 9 2024 17:00 + 8h ->
    March 10 2024 01:00

March 9 2024 17:00 + 9h ->
    March 10 2024 03:00

March 9 2024 17:00 + 10h ->
    March 10 2024 04:00



https://xkcd.com/2867/

OBLIGATORY SUMMARY SLIDE

OBLIGATORY SUMMARY SLIDE

- COMPUTERS THINK ABOUT DATES AND TIME AS NUMERIC OFFSETS FROM A SPECIFIC POINT IN TIME

OBLIGATORY SUMMARY SLIDE

- COMPUTERS THINK ABOUT DATES AND TIME AS NUMERIC OFFSETS FROM A SPECIFIC POINT IN TIME
- HUMANS THINK ABOUT DATES IN A LOT OF DIFFERENT WAYS

OBLIGATORY SUMMARY SLIDE

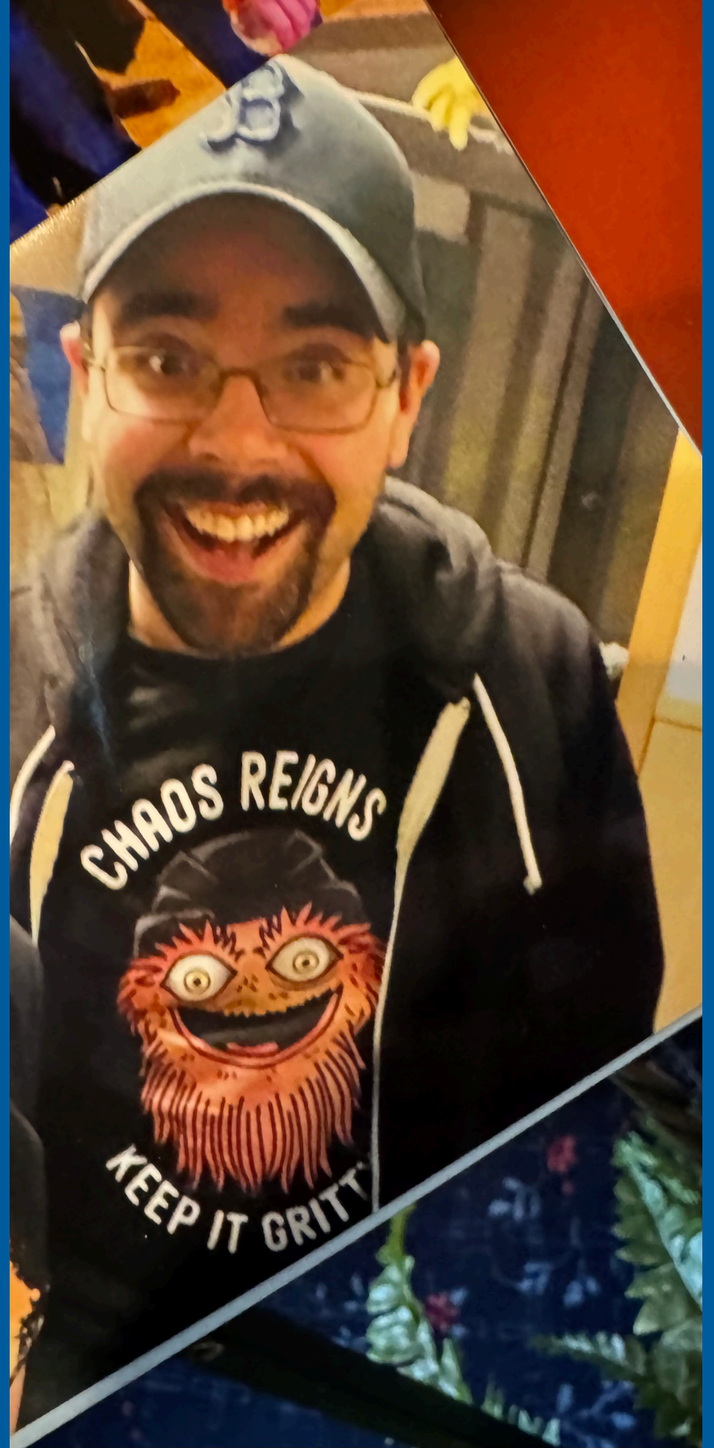
- COMPUTERS THINK ABOUT DATES AND TIME AS NUMERIC OFFSETS FROM A SPECIFIC POINT IN TIME
- HUMANS THINK ABOUT DATES IN A LOT OF DIFFERENT WAYS
- LET 🍎 HANDLE AS MUCH OF THIS FOR YOU AS POSSIBLE

OBLIGATORY SUMMARY SLIDE

- COMPUTERS THINK ABOUT DATES AND TIME AS NUMERIC OFFSETS FROM A SPECIFIC POINT IN TIME
- HUMANS THINK ABOUT DATES IN A LOT OF DIFFERENT WAYS
- LET 🍎 HANDLE AS MUCH OF THIS FOR YOU AS POSSIBLE
- USE DATE AND TIME STYLES RATHER THAN SPECIFIC FORMATS IF YOU'RE DISPLAYING A DATE TO A USER

OBLIGATORY SUMMARY SLIDE

- COMPUTERS THINK ABOUT DATES AND TIME AS NUMERIC OFFSETS FROM A SPECIFIC POINT IN TIME
- HUMANS THINK ABOUT DATES IN A LOT OF DIFFERENT WAYS
- LET 🍎 HANDLE AS MUCH OF THIS FOR YOU AS POSSIBLE
- USE DATE AND TIME STYLES RATHER THAN SPECIFIC FORMATS IF YOU'RE DISPLAYING A DATE TO A USER
- TEST THE CRAP OUT OF TRANSITION POINTS


CHAOS REIGNS
KEEP IT GRITTY

DONATION LINKS

🇬🇧 https://www.thebraintumourcharity.org/donate/

🇺🇸 https://glioblastomafoundation.org/get-involved/donate

THANK YOU!

DATE AND TIME LINKS!

- HTTPS://DEVELOPER.APPLE.COM/VIDEOS/PLAY/WWDC2020/10160/ - APPLE'S GUIDE TO DISPLAYING CONTENT HUMANS UNDERSTAND ACROSS THE WORLD
- HTTPS://WWW.YOUTUBE.COM/WATCH?V=-5WPM-GESOY - TOM SCOTT FROM 2013 ON THE PROBLEM WITH TIME AND TIME ZONES. CAPTURING THE EXASPERATION BEAUTIFULLY
- HTTPS://GOSHDARNFORMATSTYLE.COM/ - A GUIDE TO SWIFTUI `FormatStyle` FUNTIMES

LINKS:  TIME EDITION

- https://www.esa.int/Applications/Satellite_Navigation/Telling_time_on_the_Moon. The initial proposal the ESA made for coming up with a lunar time zone
- https://www.smithsonianmag.com/smart-news/the-moon-will-get-its-own-time-zone-called-coordinated-lunar-time-under-nasas-lead-180984076/. A very readable summary of the effort that will be led by

LINKS: SERIOUSLY. LISTEN TO DAVE DELONG EDITION

- HTTPS://YOURCALENDRICALFALLACYIS.COM, A FINE LISTING OF JUST A FEW OF THE MANY. MANY. MANY WAYS TO SCREW UP DATES. TIMES. AND CALENDARS.
- HTTPS://VIMEO.COM/865876497. HIS 2023 NSSPAIN TALK "THE TEMPORAL AXIS OF SPACE-TIME" DRAWING PARALLELS BETWEEN HOW WE THINK ABOUT SPACE AND HOW WE THINK ABOUT TIME