

# Dates and Times with Their Context

Resolve repeated hours, preserve calendar meaning and  
reconcile elapsed time

01

---

---

02

---

---

03

---

---

# How to use this guide

People exchanging schedules and timestamped records across regions.

An original overnight workshop case uses six timestamp records, a repeated local hour, a skipped hour and a ninety-minute session to make temporal assumptions explicit.

- Separate date, local time, offset, named zone and elapsed duration.
- Preserve source timestamps while producing inspectable UTC views.
- Classify ambiguous, nonexistent and unique local times with packaged IANA rules.
- Reconcile a ninety-minute session and reject unsupported timestamp forms explicitly.

## Scope and limitations

- The parser supports a declared subset rather than every RFC 3339 representation.
- Zone-dependent examples require Python 3.9+ and installed tzdata; no automatic installation occurs.
- Historical fixtures do not guarantee future time-zone rules, calendar-service behavior or precision timekeeping.

Published by KitForma. Edition 1.0. Sources checked 1 October 2026. People and businesses in worked examples are fictional. This is an independent publication, not endorsed by the platforms mentioned.

Copyright 2026 KitForma. Usage, support and refund terms are included in LICENSE.txt in the purchased package.

# Contents

|                                               |   |
|-----------------------------------------------|---|
| How to use this guide                         | 2 |
| Contents                                      | 3 |
| Ask what kind of time value you actually have | 4 |
| Sources and tools                             | 6 |

# Ask what kind of time value you actually have

The fictional Harbour Practice Desk is coordinating an overnight document workshop with participants in several regions. Its records include a workshop date, a local meeting time, timestamps with numeric offsets, and an elapsed session duration. These values look similar in a spreadsheet but do not mean the same thing. Your first task is to classify them before converting anything. A convenient format can conceal a serious mistake if a calendar date is silently treated as an instant.

A date names a day in a calendar context. A wall-clock time describes what a local clock displays. A numeric offset relates a particular local timestamp to UTC. A named time zone supplies rules for how that relationship can vary over time. A duration measures an interval under a stated interpretation. The case uses these distinctions to explain why two entries reading 01:30 can refer to different instants and why a session can last longer than a simple subtraction of clock labels suggests.

The supplied fixtures contain six explicit timestamp records, three unresolved local-time cases, one date-only record, and several parser rejection examples. A local Python checker converts the supported timestamps, consults a packaged IANA time-zone database for Europe/London, and verifies the expected classifications. It does not create calendar events, contact participants, alter a device clock, or claim to have tested a scheduling service. The examples are fictional even though they use real time-zone rules.

Write the business question beside each field. For a workshop date, the question might be which day appears on the printed program. For an arrival log, it might be which event happened first. For a recurring meeting,

it might be what local time the organizer intends each week. Different questions require different stored context. Do not normalize every field to midnight UTC simply because one database column makes that easy; normalization should preserve meaning rather than erase it.

# Sources and tools

## [S1] RFC 3339: Date and Time on the Internet

<https://www.rfc-editor.org/rfc/rfc3339.html>

Checked: 2026-10-01

## [S2] IANA: Time Zone Database

<https://www.iana.org/time-zones>

Checked: 2026-10-01

## [S3] Python: zoneinfo, data sources and fold

<https://docs.python.org/3/library/zoneinfo.html>

Checked: 2026-10-01

## [S4] Python: datetime types and arithmetic

<https://docs.python.org/3/library/datetime.html>

Checked: 2026-10-01

## Inside the complete guide

Preserve the original before producing a normalized value

Use offsets for instants without confusing them with zones

Resolve the hour that happens twice

Recognize the local time that never happened

Compute elapsed duration on a consistent timeline

Sort records by their meaning rather than their typography

Keep zone data and compatibility part of the evidence

Reject uncertain inputs with an actionable reason

Capstone: explain a ninety-minute session across a repeated hour