

COURSE

Time: Clocks, Calendars and How We Measure It

12 Chapters

Philip Osztromok

Generated with Claude

Contents

Chapter 1	Why Time Needs Measuring	2
Chapter 2	Shadows and Water	9
Chapter 3	Moon and Sun Calendars	18
Chapter 4	The Lost Days	27
Chapter 5	Weeks, Months, Eras	36
Chapter 6	The Mechanical Clock	45
Chapter 7	Pendulums and Springs	51
Chapter 8	Longitude at Sea	58
Chapter 9	Standard Time	67
Chapter 10	Quartz and Atoms	74
Chapter 11	Computer Time	82
Chapter 12	Strange Time	90

Time: Clocks, Calendars and How We Measure It 🏆

Course 1 · Chapter 1 · The first problem of timekeeping

Imagine you are a hunter-gatherer living some forty thousand years ago, and you want to meet your cousins at the river when the salmon run starts. You cannot send a text, there is no calendar on the wall, and nobody has invented a number for "next month". What you have is the sky. The Sun comes up and goes down. The Moon grows fat, thins away and returns. The seasons swing round and the cold comes back. These three rhythms are the oldest clocks there are, and for almost the whole of human history they were the only ones. This course follows the long story of how people learned to count them, argue about them, build machines to imitate them, and finally replace them with the vibrations of atoms.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The astronomical constants (day, month and year lengths) are the standard published values, written here from memory without looking them up again during writing. The descriptions of the ancient bones are summaries of what archaeologists have proposed, and the lunar readings of them are interpretations, not settled facts.

Three Natural Clocks

Each natural clock comes from a different motion of the Earth or the Moon, and that is the root of every calendar headache in the chapters to come.

CYCLE	WHAT IS REALLY MOVING	WHAT A PERSON SEES	LENGTH
Day	the Earth spins on its axis	sunrise to sunrise	24 hours (mean solar day)
Month	the Moon orbits the Earth	new Moon to new Moon	about 29.53 days (synodic month)
Year	the Earth orbits the Sun	one full round of the seasons	about 365.2422 days (tropical year)

The Earth spinning, the Moon circling and the Earth circling the Sun are three separate motions. Nothing in physics makes one of them a tidy multiple of another. As the next section shows, the numbers do not fit.

Why They Do Not Divide Evenly

How many months are in a year? If the answer were a whole number, calendar-making would be easy. Divide the year by the month:

```
YEAR = 365.2422            # tropical year, in days
MONTH = 29.530589          # synodic month (new Moon to new Moon), in days

print(round(YEAR / MONTH, 3))
print(round(12 * MONTH, 3), round(YEAR - 12 * MONTH, 3))
print(round(13 * MONTH, 3), round(13 * MONTH - YEAR, 3))
# 12.368
# 354.367 10.875
# 383.898 18.655
```

A year contains **12.368** Moon cycles: twelve and a bit. Twelve months add up to 354.367 days, which is 10.875 days short of a year. Thirteen months overshoot by 18.655 days. So a calendar built purely on the Moon slides backwards through the seasons by nearly eleven days a year, and a calendar built purely on the Sun ignores the Moon completely. Every calendar in history is a compromise between these two facts, and the compromises fill chapters 3 and 4.

The day has the same trouble with the year. The year is 365.2422 days, so if you count days you must carry that extra 0.2422 of a day (nearly six hours) somewhere. A calendar of exactly 365 days loses about a quarter of a day each year. After four years it is nearly a whole day (0.97 of a day) behind the seasons.

```
Year       365.2422 days
12 months  354.367 days  (short by 10.875)
13 months  383.898 days  (over by 18.655)
365 days   short by 0.2422 per year  (about 5 h 49 min)
```

A Closer Look at the Day

Even "a day" is less simple than it sounds. There are two sensible ways to measure one, and they give different answers.

	SOLAR DAY	SIDEREAL DAY
Measured	from noon to noon (the Sun back at its highest)	from one passage of a fixed star to the next
Length	24 h 0 min 0 s on average	about 23 h 56 min 4.09 s
Used for	daily life, civil time	astronomy, pointing telescopes

Why the difference? While the Earth spins once, it also moves a little along its orbit around the Sun. After one full turn relative to the stars the Earth has to turn a little further for the Sun to be back at its highest. Over a whole year, this adds up to exactly one extra turn of the Earth, so a year contains one more sidereal day than solar days: 366.2422 sidereal days, against 365.2422 solar days. The sidereal day is the solar day scaled by that ratio:

```
YEAR = 365.2422
solar_day = 86400.0                    # seconds
sidereal_days_per_year = YEAR + 1        # one extra turn per orbit
sidereal_day = YEAR * solar_day / sidereal_days_per_year

h, rem = divmod(sidereal_day, 3600)
m, s = divmod(rem, 60)
print(round(sidereal_day, 2))
print(int(h), int(m), round(s, 2))
print(round(solar_day - sidereal_day, 2))
# 86164.09
# 23 56 4.09
# 235.91
```

The sidereal day is **86,164.09 seconds**, or 23 h 56 min 4.09 s, which is 235.91 seconds (about 3 min 56 s) shorter than the solar day. That is why a given star rises about four minutes earlier each night, and why different constellations dominate the sky in different seasons.

Neither day is perfectly constant. The solar day varies slightly in length through the year (chapter 2 explains the equation of time), and tides slowly drag on the Earth's spin so that the day is getting longer by roughly a couple of milliseconds per century. For now, take the "24 hours" day as an average.

The Oldest Records: Notches on Bone

How early did people count these cycles? We cannot know what they said or thought, but some objects look very much like tallies. The most famous are two bones from Africa:

- **The Lebombo bone**, from a cave in the Lebombo mountains on the border of South Africa and Eswatini, is a piece of baboon leg bone about 43,000 years old, with about 29 clear notches cut in it. The bone is broken at one end, so there may once have been more. A count near 29 is close to a lunar month, which is why it is sometimes called a lunar tally.
- **The Ishango bone**, found in what is now the Democratic Republic of the Congo, is roughly 20,000 years old and carries groups of notches in three columns. It has been read as a tally, as a lunar record and even as a simple arithmetic aid.

Be careful with these claims. Marks on bone could be a hunting tally, a trade record or decoration, and nobody can prove which. What is safe to say is this: tens of thousands of years ago, people made patterned marks in sets that look like counting, and counting the Moon is one reasonable explanation. A tally keeps one thing the memory cannot: a number you can check tomorrow.

Tally Drift: Why Counting Is Not Enough

Suppose a bone-carver cuts one notch per day and starts a new column at each new Moon. They must decide how long a month is. Real months last 29 or 30 days, never both at once. Counting every month as 29 days makes the tally lag the Moon. Counting 30 makes it run ahead. How quickly?

```
MONTH = 29.530589

def drift_days(months, tally_length):
    """Days by which a fixed-length tally has drifted from the Moon."""
    return months * (MONTH - tally_length)

for length in (29, 30, 29.5):
    print("tally", length, "-> drift after 12 months:", round(drift_days(12, length), 3)
# tally 29 -> drift after 12 months: 6.367 days
# tally 30 -> drift after 12 months: -5.633 days
```

```
  # tally 29.5 -> drift after 12 months: 0.367 days
```

A fixed 29-day month falls 6.4 days behind the Moon in a year. A fixed 30-day month runs 5.6 days ahead. Even the best fixed guess, 29.5 days (alternating 29 and 30), is still 0.367 days out after twelve months and a full day out after about 33 months. A tiny error in a repeated count does not stay tiny: it adds up. This is the key idea of the whole course, and we will meet it again with calendars, with pendulum clocks, with ships at sea and with computers. **A timekeeper is judged less by how well it starts than by how its error grows.**

What Makes a Good Clock?

We can now ask what we mean by a "good" clock. People use three words loosely, but they mean different things. Definitions vary a little between fields. These are the working meanings used in this course.

WORD	QUESTION IT ANSWERS	EXAMPLE OF FAILING
Accuracy	How close is the clock to the true time, on average?	a clock that runs 10 minutes fast
Precision	How finely can it be read, and how repeatable is the reading?	a clock with only an hour hand
Stability	How steady does its rate stay from one day to the next?	a clock gaining 1 s one day and losing 3 s the next

A very *stable* clock is the most useful, even if it is inaccurate, because a steady error can be measured once and corrected for ever. A stable clock that gains exactly 2 seconds a day can be set right by subtracting 2 seconds every day. An unstable clock cannot be corrected in advance. This is why later chapters spend so much effort on making the rate steady (pendulums, balance springs, quartz, atoms) rather than on setting the starting time.

Errors are often quoted as a fraction. A clock that gains 1 second in a day is wrong by 1 part in 86,400. Engineers write this as **parts per million (ppm)**:

```
SECONDS_PER_DAY = 86400
for err in (1, 2, 10):
```

```
    ppm = err / SECONDS_PER_DAY * 1_000_000
    per_year = err * 365
    print(err, "s/day =", round(ppm, 2), "ppm =", per_year, "s/yr =", round(per_year / 6
# 1 s/day = 11.57 ppm = 365 s/yr = 6.1 min/yr
# 2 s/day = 23.15 ppm = 730 s/yr = 12.2 min/yr
# 10 s/day = 115.74 ppm = 3650 s/yr = 60.8 min/yr
```

A clock only 1 second a day out looks wonderful, yet it is 6 minutes a year out, and 10 seconds a day means an hour a year. For navigation, science and the internet, these small errors are not small at all. By the end of the course you will see clocks that are wrong by less than one second in many millions of years.

Common mistakes

Treating "a year" as 365 days exactly (it is 365.2422, which is the reason leap years exist); assuming 12 months make a year (12 Moon cycles are 10.875 days short); confusing the solar day with the sidereal day (they differ by about 4 minutes); and mixing up accuracy with stability (a clock can be badly wrong yet perfectly steady, or right on average yet jumpy). Another common slip: reading a tally bone as proof of a specific ancient calendar. Treat those readings as plausible interpretations.

EXERCISE 1

Using a year of 365.2422 days and a month of 29.530589 days, find how many months are in a year, and how far 12 and 13 months miss a year. Then search for the whole number of years (up to 39) whose length is closest to a whole number of months, printing each new best fit. Which one wins, and by how many days is it off?

View solution

EXERCISE 2

Three clocks were compared with the true time at midnight on 8 days, giving the offsets listed in the solution file. For each clock, find the mean daily rate (a measure of accuracy) and the spread of the daily changes (a measure of stability) using Python's `statistics` module. Which clock is the easiest to improve, and why?

View solution

EXERCISE 3

A tally maker cuts one notch per day. Using a Moon month of 29.530589 days, find how far the tally has drifted after 12 months if every month is counted as 29, 30 or 29.5 days, and after how many months each version is a whole day out. Then simulate alternating 29- and 30-day months and print the running error for 12 months.

View solution

QUICK REFERENCE

- **Three natural clocks:** day (Earth spins), month (Moon orbits), year (Earth orbits the Sun)
- **Lengths:** mean solar day 24 h; synodic month 29.530589 days; tropical year 365.2422 days
- **They do not fit:** a year is 12.368 months, so 12 months fall 10.875 days short
- **Sidereal day:** 86,164.09 s (23 h 56 min 4.09 s), about 236 s shorter than the solar day
- **Drift:** a small error repeated many times adds up (1 s/day is 6 min/yr)
- **Accuracy / precision / stability:** closeness to true time / fineness of reading / steadiness of rate

COMING NEXT

Time: Clocks, Calendars and How We Measure It 2 is about shadows and water: how a stick in the ground became a sundial, why the Sun is not a perfect clock, and how water clocks kept time when the Sun was hidden.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 2 · The first machines for the hours

Tally marks count days, but they cannot tell you whether it is nearly noon. For that, people needed to read the time within a single day, and the first tool was already lying on the ground: a shadow. Stick a post upright in flat ground and watch it. The shadow swings round through the day, shrinking towards noon and stretching again towards evening. Time: Clocks, Calendars and How We Measure It 1 showed that the day is the steadiest of the natural clocks. This chapter shows how people cut it into pieces, first with the Sun and then, when the Sun was hidden or gone, with water, fire and smoke.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The sundial and day-length formulas are standard textbook ones; the equation of time uses a common approximation that is good to roughly a minute, not the exact value. The dates for ancient devices (Egyptian shadow clocks and water clocks, Greek inventions, Chinese incense clocks) are the approximate dates usually given and were written from memory, not looked up again while writing. Treat them as "around then", not as precise years.

The Gnomon: a Stick That Tells Time

A **gnomon** (Greek for "one who knows") is any object that casts a shadow to be read: a stick, a post, a tall stone. It gives you two different pieces of information, and people used both.

- **The direction of the shadow** tells the time of day. In the morning the shadow points west, at noon it points toward the nearest pole (north in the northern hemisphere), and in the evening it points east. The shadow is shortest at local noon, so a shadow marked at its shortest gives you a north-south line.
- **The length of the shadow** tells the season. The Sun climbs high in summer and stays low in winter, so the same noon shadow is short in June and long in December.

The link between height and shadow is plain school geometry. If the Sun is at an angle (altitude) above the horizon, a gnomon of height *h* casts a shadow of length *h* divided by tan(altitude). At noon the altitude is 90 degrees minus your latitude, plus the Sun's declination (how far north or south of the equator the Sun is overhead, which is +23.44 degrees at the June solstice, 0 at the

equinoxes and -23.44 at the December solstice). Try it for a 1 metre stick at latitude 56 degrees north, roughly Edinburgh:

```
import math

GNOMON = 1.0                      # height of the stick, metres
LAT = 56.0                        # roughly Edinburgh, degrees north
SUN_DECLINATION = {"June solstice": 23.44, "Equinox": 0.0, "December solstice": -23.44}

for name, dec in SUN_DECLINATION.items():
    noon_altitude = 90 - LAT + dec                # Sun's height above the horizon at noo
    shadow = GNOMON / math.tan(math.radians(noon_altitude))
    print(name, round(noon_altitude, 2), "deg, shadow", round(shadow, 2), "m")
# June solstice 57.44 deg, shadow 0.64 m
# Equinox 34.0 deg, shadow 1.48 m
# December solstice 10.56 deg, shadow 5.36 m
```

The noon shadow of a 1 metre stick is 0.64 m in midsummer and 5.36 m in midwinter, more than eight times longer (5.36 / 0.64 = 8.4). A long shadow like that is how an ancient observer could find the solstices without any calendar: watch for the day the noon shadow stops getting shorter or longer. At the equinoxes the noon shadow is 1.48 m. That is not the average of the two extremes (which would be 3.0 m), because shadow length depends on the tangent of the angle, not on the angle itself.

From Stick to Sundial

A *shadow clock* or *sundial* adds markings so that the shadow reads the hour directly. Egyptian shadow clocks of around 1500 BC were T-shaped or stepped blocks that were turned to face east in the morning and west in the afternoon. Later dials used a flat plate with hour lines. The Earth turns once in 24 hours, which is 15 degrees every hour (360 / 24 = 15), so the shadow on a dial built to point at the pole moves 15 degrees per hour.

There is one important design detail. On a flat horizontal dial the shadow does not move at a steady rate unless the gnomon is tilted to point at the celestial pole, parallel to the Earth's axis. That means the angle of the gnomon above the dial equals your latitude: about 56 degrees in

Edinburgh, 30 in Cairo. This is why a sundial made for one town reads wrongly in another far to the north or south: the gnomon angle and the hour lines were made for one latitude.

```
N (towards the pole star)
|
|      gnomon: tilted up from the plate by your latitude,
|    /         so it points along the Earth's axis
|  /
|/_ _ _ _ _ _ _ _  dial plate (flat and level)
S

side view: angle between gnomon and plate = latitude
```

The Sun Is Not a Perfect Clock: the Equation of Time

A sundial seems like the truest clock, since it follows the Sun itself. But it disagrees with a steady clock by up to a quarter of an hour, and the amount changes through the year. The gap is called the **equation of time**: apparent solar time (the sundial) minus mean solar time (an evenly running clock).

There are two causes, and both have to do with the shape and tilt of the Earth's path.

- **The orbit is an ellipse.** The Earth moves faster when closer to the Sun (early January) and slower when farther away (early July). The Sun therefore appears to drift east against the stars faster in winter, so the Earth must turn a little more each day to bring the Sun back to its highest point, and the solar day is a little longer than average.
- **The axis is tilted.** The Sun's yearly path (the ecliptic) is tilted against the equator, so part of its apparent drift is wasted going up or down rather than along the direction the Earth turns. The effect repeats twice a year.

The two effects add up to a curve with two highs and two lows. A simple formula, good to about a minute, reproduces it:

```
import math

def equation_of_time(day_of_year):
```

```
    """Approximate equation of time in minutes (sundial minus clock)."""
    b = math.radians(360 / 365 * (day_of_year - 81))
    return 9.87 * math.sin(2 * b) - 7.53 * math.cos(b) - 1.5 * math.sin(b)

for label, day in (("11 Feb", 42), ("14 May", 134), ("26 Jul", 207), ("3 Nov", 307)):
    print(label, round(equation_of_time(day), 1), "min")
# 11 Feb -14.6 min
# 14 May 3.8 min
# 26 Jul -6.2 min
# 3 Nov 16.4 min
```

A positive number means the sundial is ahead of the clock. On 3 November it is about 16 minutes ahead, so the Sun is highest at about 11:44 by an evenly running clock (ignoring your longitude within the time zone). On 11 February it is about 15 minutes behind, and the Sun is highest at about 12:15. The total swing from one extreme to the other is about 31 minutes. Four days a year (near mid April, mid June, the end of August and just before Christmas) sundial and clock agree. Sundial makers often engraved a small correction table, or a figure-of-eight curve called an *analemma*, on the dial for this reason.

This is the first place in the course where a "natural" clock turns out not to be uniform. A clock that simply follows the Sun is accurate over a year but not *stable* from day to day in the sense used in Time: Clocks, Calendars and How We Measure It 1. When mechanical clocks arrived (Time: Clocks, Calendars and How We Measure It 6), people had to decide which one was correct: the steady machine or the heavens. The machine won, and "mean time" became the standard.

Unequal Hours and Equal Hours

Our hour is a fixed length: 60 minutes, all year round. For most of history this was not so. Egyptians, Greeks, Romans and medieval Europeans commonly divided the *daylight* into 12 hours and the *darkness* into another 12. These **seasonal** or **unequal hours** were short in winter days and long in summer days. A sundial can show them directly, because the shadow's position between sunrise and sunset gives the fraction of the day gone. Let us see how much they stretch:

```
import math
```

```
def daylight_hours(lat, declination):
    """Sunrise to sunset, ignoring atmospheric refraction."""
    half = math.degrees(math.acos(-math.tan(math.radians(lat)) * math.tan(math.radians(d
    return 2 * half / 15

for lat in (56, 30):
    for name, dec in (("June", 23.44), ("Equinox", 0.0), ("December", -23.44)):
        day = daylight_hours(lat, dec)
        day_hour = day * 60 / 12
        night_hour = (24 - day) * 60 / 12
        print(lat, name, round(day, 2), "h daylight | day hour", round(day_hour, 1), "mi
# 56 June 17.33 h daylight | day hour 86.7 min | night hour 33.3 min
# 56 Equinox 12.0 h daylight | day hour 60.0 min | night hour 60.0 min
# 56 December 6.67 h daylight | day hour 33.3 min | night hour 86.7 min
# 30 June 13.93 h daylight | day hour 69.7 min | night hour 50.3 min
# 30 Equinox 12.0 h daylight | day hour 60.0 min | night hour 60.0 min
# 30 December 10.07 h daylight | day hour 50.3 min | night hour 69.7 min
```

At Egypt's latitude (30 degrees north) a daytime hour swings between about 50 and 70 minutes. At Edinburgh's latitude the swing is huge: an hour in a June day lasts 87 minutes, and in December just 33 (17.33 hours of June daylight divided by 12 is 1.444 hours, which is 86.7 minutes). A daily routine such as "work until the sixth hour" therefore floats with the seasons, while the *number* of hours stays fixed at 12. That made such hours convenient for ordinary life: whatever the season, the sixth hour is always midday.

	UNEQUAL (SEASONAL) HOURS	EQUAL HOURS
Definition	daylight / 12 and darkness / 12	1/24 of a whole day, always
Length	changes through the year and with latitude	always 60 minutes
Best for	daily life and schedules tied to the Sun	astronomy, later mechanical clocks and travel
Needs	a clock whose rate changes through the year	a clock that runs at a single steady rate

Astronomers used equal hours early, since they needed constant units to compare observations. Everyday life used unequal hours for a very long time, and equal hours spread widely once clocks that ran at one rate became common (the first mechanical clocks are the subject of Time: Clocks, Calendars and How We Measure It 6).

Water Clocks: Time Without the Sun

A sundial is useless at night, in cloud, or indoors. The answer, from Egypt and Mesopotamia onward, was to let something else flow at a steady rate and measure what had flowed. The **water clock** (Greek *klepsydra*, "water thief") is the oldest clock that works anywhere. The oldest surviving example is an Egyptian alabaster vessel from the Karnak temple, from roughly 1400 BC, usually dated to the reign of Amenhotep III. It was a tall bowl with a small hole near the bottom and hour marks on the inside, so the falling water level showed the time.

There are two designs. An *outflow* clock drains from a vessel, and the water level in the vessel (or the amount collected below) is the time. An *inflow* clock fills a vessel, and a float rising inside carries a pointer. Both have the same problem: **water does not leave at a steady rate**. The deeper the water, the greater the pressure and the faster it runs out. In physics this is Torricelli's law, from the 17th century: the speed of the jet is proportional to the square root of the depth. Let us watch what happens in a plain cylindrical tank with a small hole:

```
import math

G = 9.81            # gravity, m/s^2
AREA_TANK = 0.01    # tank cross-section, m^2
AREA_HOLE = 0.00001 # hole, m^2 (0.1 cm^2)
COEFF = 0.6         # typical loss factor for a small hole
level = 0.30        # water depth at the start, metres
t = 0
next_mark = 0.25
last_time = 0

while level > 0.05:
    outflow = COEFF * AREA_HOLE * math.sqrt(2 * G * level)   # m^3 per second
    level -= outflow / AREA_TANK                              # one-second step
    t += 1
    if level <= next_mark:
        print("level", round(next_mark * 100), "cm at", t, "s (this 5 cm took", t - last
        last_time = t
        next_mark = round(next_mark - 0.05, 2)
# level 25 cm at 36 s (this 5 cm took 36 s)
# level 20 cm at 76 s (this 5 cm took 40 s)
# level 15 cm at 121 s (this 5 cm took 45 s)
# level 10 cm at 174 s (this 5 cm took 53 s)
```

```
  # level 5 cm at 244 s (this 5 cm took 70 s)
```

The first 5 cm of fall takes 36 seconds and the last 5 cm takes 70, nearly twice as long (70 / 36 = 1.94). If hour marks were spaced evenly on the inside, the clock would run slow as it emptied. Ancient makers worked out two cures by trial. One was to shape the vessel so the level falls at a roughly steady rate (the sloping sides of the Karnak bowl are generally thought to serve this purpose; Exercise 3 tries it). The other, used by the Greeks and later, was a *float regulator*: an upper tank kept topped up so that the pressure over the outlet stayed constant. Ctesibius, who worked in Alexandria in the third century BC, is credited with a version of this and with adding gears and a pointer. Water clocks give us the first example of a theme that returns again and again: a clock is only as good as the steadiness of its rate.

Water clocks were not only for amusement. Greek and Roman courts used them to limit the length of speeches, and a famous octagonal building in Athens, the Tower of the Winds (first century BC), is usually described as combining sundials with a water clock. Water also freezes in winter and flows differently with temperature (warm water is runnier), so water clocks were never as reliable as one would like.

Candles, Incense and Burning Time

Fire gives a second kind of steady flow. A candle burns down at a fairly constant rate, so marks along its side can be read as hours. Suppose a candle burns 2.5 cm each hour. A 30 cm candle then lasts 30 / 2.5 = 12 hours, and a mark every 2.5 cm gives hourly readings. Candle clocks are described in medieval Europe and in the Islamic world, and Chinese sources mention them early. An alarm version is often described: a metal ball held by a pin pushed into the candle at the right depth would drop into a dish with a clang when the flame reached the pin.

Incense clocks are a Chinese and Japanese variant. A fine incense powder is pressed into a groove, often a winding pattern in a wooden or metal tray, and lit at one end. The glowing tip burns steadily along the groove. Different scents could be laid down at different points, so that a change of smell announced the hour. Such clocks were in use in East Asia for many centuries.

CLOCK	WHAT FLOWS OR BURNS	WORKS IN THE DARK?	MAIN WEAKNESS
Sundial	the Sun's shadow	no	useless at night or in cloud; wrong for the wrong latitude
Water clock	water through a hole	yes	flow depends on depth and on temperature; freezes
Candle clock	wax burning down	yes	draughts and uneven wax change the rate
Incense clock	a glowing trail of powder	yes	damp and draughts; one use per fill

Notice the pattern. None of these clocks is accurate to a minute. Each is good enough for its job (when to open a market, when to change a guard, when a speaker must stop), and each depends on something that flows at a roughly steady rate. The next big step was not to find a better substance but to find a better *rhythm*, and that story begins with the calendars of Time: Clocks, Calendars and How We Measure It 3 before it reaches machines.

Common mistakes

Assuming a sundial agrees with a watch (it can be 16 minutes out, and the gap changes through the year); thinking the gnomon on a horizontal dial can be set at any angle (it must equal the latitude); forgetting that the equation of time has two highs and two lows, not one; believing an "hour" always lasted 60 minutes; and treating the water level in a plain cylinder as falling at a constant rate (it slows as it empties). Another slip is to take "oldest surviving" to mean "first invented": earlier clocks may simply not have survived.

EXERCISE 1

For a 1 metre gnomon, find the noon shadow length at the June solstice, the equinox and the December solstice for latitudes 30 and 56 degrees north. Then work out the length in minutes of a "day hour" and a "night hour" on each of those dates. Which latitude has the bigger swing in shadow length, and the bigger swing in hour length?

View solution

EXERCISE 2

Using the equation of time function in this chapter, scan all 365 days to find the day the sundial is furthest ahead of the clock and the day it is furthest behind. Print the total swing and the days where the sign changes. Then work out, in hours, minutes and seconds, the clock time at which the Sun is highest on 3 November and on 11 February.

View solution

EXERCISE 3

Simulate the leaking water clock from this chapter and print the seconds each 5 cm of fall takes, together with the ratio of the last interval to the first. Then change the tank so that its cross-section grows with the square root of the depth, and compare. What does this change do to the readings?

View solution

QUICK REFERENCE

- **Gnomon:** direction of the shadow gives the hour, length gives the season; noon shadow = height / tan(90 - latitude + declination)
- **Sundial:** the Earth turns 15 degrees per hour; the gnomon angle must equal the latitude
- **Equation of time:** sundial minus clock, about +16 min (early Nov) to -15 min (mid Feb), a 31 minute swing from the Earth's tilt and elliptical orbit
- **Unequal hours:** daylight / 12; at 56 degrees north a day hour is 87 min in June, 33 in December
- **Water clock:** flow slows as the level falls (first 5 cm took 36 s, last took 70 s); fixed by shaping the vessel or a constant-pressure regulator
- **Candle and incense clocks:** burn at a steady rate; useful in the dark and indoors

COMING NEXT

Time: Clocks, Calendars and How We Measure It 3 is about calendars built on the Moon, the Sun, or both: how the Egyptians, Babylonians, Hebrews and Muslims fitted months into years, and why a year is 365.2422 days.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 3 · Fitting months into years

Imagine you are a temple scribe and the harvest festival is meant to fall in the first full Moon after the barley ripens. Count 12 Moon cycles from last year's festival and you arrive about eleven days too early; the barley is still green. Count 13 and you are over a fortnight too late. Time: Clocks, Calendars and How We Measure It 1 put numbers on this clash: a year holds 12.368 months, and no whole number of months fits. This chapter is about the three ways people have answered. Follow the Moon and let the seasons wander, follow the Sun and forget the Moon, or stitch the two together with an occasional extra month.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The year, month and 19-year figures are standard values and were calculated in the text. The historical dates (the Egyptian civil calendar, Babylonian regularisation of the 19-year cycle, the fixing of the Hebrew calendar, the start of the Islamic era in 622 CE) are the approximate dates usually given and were written from memory, not looked up again while writing. Treat them as "around then" for the ancient ones. The Hebrew leap-year pattern and the Islamic 30-year pattern are the commonly published ones, but the real calendars have extra rules for month lengths and for the day a year begins that this chapter leaves out.

Three Ways to Build a Calendar

Every calendar has to decide what it is tied to. There are three families, and which one a culture chose still shows in its festivals today.

	LUNAR	SOLAR	LUNISOLAR
Tied to	the Moon's phases	the seasons	both
Typical year	12 months, 354 or 355 days	about 365 days	12 or 13 months, 353 to 385 days
Months match the Moon?	yes	no, months are just a convenient division	yes

	LUNAR	SOLAR	LUNISOLAR
Year matches the seasons?	no, slides through them	yes (once leap days are handled)	yes, on average
Examples	Islamic calendar	Egyptian civil calendar, and the calendars of Time: Clocks, Calendars and How We Measure It 4	Babylonian, Hebrew, Chinese

Lunar Calendars: Follow the Moon, Lose the Seasons

The simplest way to run a month is to let the sky tell you. A new month begins when someone sees the thin crescent of the new Moon after sunset. Because the Moon cycle is 29.53 days, a month is either 29 or 30 days: whichever the sighting says. Twelve such months give a year of 354 or 355 days. That is the whole design, and it works beautifully for the Moon. It does not work for the seasons, as a small calculation shows.

```
YEAR = 365.2422
MONTH = 29.530589

lunar_year = 12 * MONTH
slip = YEAR - lunar_year
print("12 lunar months:", round(lunar_year, 3), "days")
print("the seasons get ahead by", round(slip, 3), "days a year")
print("a full lap of the seasons takes", round(YEAR / slip, 1), "years")
# 12 lunar months: 354.367 days
# the seasons get ahead by 10.875 days a year
# a full lap of the seasons takes 33.6 years
```

A purely lunar calendar slips back about 10.9 days against the seasons every year. After three years it is a month adrift (3 x 10.875 = 32.6 days), and after 33.6 years the New Year has been through summer, autumn, winter and spring and arrived back where it began (365.2422 / 10.875 = 33.6). Someone born in the first month of the calendar will celebrate birthdays in every season of their life.

The Islamic Calendar: Lunar on Purpose

The **Islamic (Hijri) calendar** is the main calendar still in use that is purely lunar, and it is lunar by choice. It counts years from the Hijra, the Prophet Muhammad's move from Mecca to Medina, in 622 CE. It has 12 months, and the Quran is generally understood to forbid adding a thirteenth, so the calendar never gets pulled back towards the seasons. The month of Ramadan therefore falls about 11 days earlier each year and, over about 33 years, occurs in every season.

In practice a month begins when the crescent is actually sighted, which depends on weather and place, so the beginning of a month can differ by a day between countries. For planning, tables use a fixed rule: the 30-year tabular calendar. Of every 30 years, 19 have 354 days and 11 have 355 days (leap years are years 2, 5, 7, 10, 13, 16, 18, 21, 24, 26 and 29 of the cycle). The arithmetic is short:

```
YEAR = 365.2422
calendars = {"Egyptian civil": 365, "12 lunar months": 354, "Islamic 30-year cycle": 106
for name, length in calendars.items():
    slip = YEAR - length
    print(name, "| slip per year", round(slip, 3), "d | after 100 years", round(slip * 1(
# Egyptian civil | slip per year 0.242 d | after 100 years 24.2 d | full lap 1508.0 years
# 12 lunar months | slip per year 11.242 d | after 100 years 1124.2 d | full lap 32.5 ye
# Islamic 30-year cycle | slip per year 10.876 d | after 100 years 1087.6 d | full lap 3
```

The 30-year cycle holds 19 x 354 + 11 x 355 = 6726 + 3905 = 10631 days, which is 354.367 days a year on average, and 10631 / 360 = 29.5306 days a month: very close to the Moon's 29.5306. Plain 354-day years, with no leap days at all, would drift 0.367 of a day a year against the Moon itself; the tabular calendar holds the Moon to within about a day in 2500 years (Exercise 3). The price of that Moon-accuracy is that the seasons are ignored on purpose. Because the Islamic year is shorter than the solar year, its number grows faster: 1447 AH began in 2025 CE, only about 1403 solar years after 622 CE.

Solar Calendars: the Egyptian Civil Year

The opposite choice is to forget the Moon and count days in a fixed pattern close to the solar year. The classic early example is the **Egyptian civil calendar**, probably in use by the third millennium BC. Its year was beautifully simple: 12 months of exactly 30 days, then 5 extra days at the end, which makes 12 x 30 + 5 = 365 days. There were no leap days, and the months no

longer followed the Moon. The calendar was tied to the Nile, whose yearly flood arrived at about the time Sirius, the brightest star, was first seen rising ahead of the Sun at dawn. That star-rising was the natural New Year marker.

Egypt also kept a lunar calendar alongside for religious festivals, so the picture was not as simple as "solar only". But the civil year was the one used by officials, and it has a built-in flaw: 365 days is 0.2422 of a day too short. The New Year moves one day earlier against the seasons about every four years.

```
YEAR = 365.2422                 # tropical year, days
CIVIL = 365                     # 12 months of 30 days + 5 extra days
for years in (4, 40, 100, 1460):
    print(years, "years: New Year has slipped", round(years * (YEAR - CIVIL), 1), "days"
print("full lap of the seasons:", round(YEAR / (YEAR - CIVIL)), "years")
# 4 years: New Year has slipped 1.0 days
# 40 years: New Year has slipped 9.7 days
# 100 years: New Year has slipped 24.2 days
# 1460 years: New Year has slipped 353.6 days
# full lap of the seasons: 1508 years
```

The slip is slow compared with a lunar calendar, so a lifetime hardly notices it: 24 days a century. But over 1500 years it adds up to the whole year, so a festival tied to the civil date would eventually have run through all the seasons. The 1460-year figure often quoted for the full lap comes from rounding the year to exactly 365.25 days, where 365 / 0.25 = 1460. With the better value, 365.2422, the lap is 1508 years. Fixing this drift needed a leap day, and that story, including the Roman and Gregorian fixes, is in Time: Clocks, Calendars and How We Measure It 4.

Lunisolar Calendars: Stitching Moon and Sun Together

The third answer keeps both: months that follow the Moon and a year that follows the seasons. Because 12 months fall short and 13 overshoot, you use 12 most years and 13 in some. The extra month is called an **intercalary** month (from the Latin for "to proclaim a day inserted"). The art is in choosing which years get it.

Babylon. The earliest Babylonian calendars decided the matter by observation and decree: priests watched the sky and the state of the crops, and the king announced when an extra month was to be added. Later the decisions became rule-based. By around 500 BC the Babylonians had settled on a 19-year pattern with 7 extra months. The same near-match was found in Greece and is often named after Meton of Athens (around 432 BC): the **Metonic cycle**. It is the 19-year fit that the exercise in Time: Clocks, Calendars and How We Measure It 1 uncovered by searching.

```
YEAR = 365.2422
MONTH = 29.530589
print("19 years =", round(19 * YEAR, 3), "days; 235 months =", round(235 * MONTH, 3), "d
print("gap:", round(235 * MONTH - 19 * YEAR, 3), "days")
# 19 years = 6939.602 days; 235 months = 6939.688 days
# gap: 0.087 days
```

Nineteen years of 12 months make 19 x 12 = 228 months. To reach 235 you need 235 - 228 = 7 extra months, so 7 of every 19 years get 13 months. Over the whole cycle the lunar and solar calendars agree to within 0.087 of a day (about two hours), so the pattern can repeat for centuries before it needs a correction. Which 7 years? A simple rule is: *add a month whenever twelve months would end more than half a month too early.* Let us try it:

```
YEAR = 365.2422
MONTH = 29.530589

offset = 0.0        # lunar year end minus solar year end, in days
for year in range(1, 20):
    months = 12
    if offset + 12 * MONTH - YEAR < -MONTH / 2:   # twelve months would end over half a m
        months = 13
    offset += months * MONTH - YEAR
    print("year", year, "months", months, "offset", round(offset, 2), "d")
# year 1 months 12 offset -10.88 d
# year 2 months 13 offset 7.78 d
# year 3 months 12 offset -3.09 d
# year 4 months 12 offset -13.97 d
# year 5 months 13 offset 4.69 d
# year 6 months 12 offset -6.19 d
```

```
  # year 7 months 13 offset 12.47 d
  # year 8 months 12 offset 1.59 d
  # year 9 months 12 offset -9.28 d
  # year 10 months 13 offset 9.37 d
  # year 11 months 12 offset -1.5 d
  # year 12 months 12 offset -12.38 d
  # year 13 months 13 offset 6.28 d
  # year 14 months 12 offset -4.6 d
  # year 15 months 13 offset 14.06 d
  # year 16 months 12 offset 3.18 d
  # year 17 months 12 offset -7.69 d
  # year 18 months 13 offset 10.96 d
  # year 19 months 12 offset 0.09 d
```

Seven years get 13 months (years 2, 5, 7, 10, 13, 15 and 18), exactly the 7 the arithmetic demanded, and the offset never strays beyond about 14 days either way before it lands on 0.09 after 19 years. A calendar like this never gets more than half a month out of step with the seasons, which is good enough to keep a spring festival in spring.

The Hebrew calendar. The Hebrew calendar uses exactly this 19-year cycle. Its leap years are the 3rd, 6th, 8th, 11th, 14th, 17th and 19th of every cycle, and the thirteenth month is added as an extra "Adar" before the usual one. Early on, the Sanhedrin in Palestine announced each year's decision from observation; the calendar was set to a fixed, calculated form in the fourth century CE (traditionally around 359 CE), so that communities far from any authority could work out the dates themselves. The Chinese calendar is another lunisolar one, with a leap month placed by an astronomical rule rather than a fixed 19-year pattern.

Notice that the Hebrew pattern (3, 6, 8, 11, 14, 17, 19) is not the same as the one our simple rule produced (2, 5, 7, 10, 13, 15, 18). Both give 7 leap years in 19. The traditional pattern spaces the extra months a little differently, and Exercise 2 compares them.

Why Is a Year 365.2422 Days?

Nothing in physics says a year has to be a certain number of days. The number is measured, not derived: the Earth's trip around the Sun and its spin on its axis are independent motions that happen to give this ratio today. The **tropical year** is the time from one March equinox to the

next (more exactly, the time for the seasons to go once round). It is about 365.2422 days, or 365 days plus 0.2422 of a day:

```
YEAR = 365.2422
SIDEREAL = 365.256363
extra = YEAR - 365
hours = extra * 24
minutes = (hours % 1) * 60
seconds = (minutes % 1) * 60
print(int(hours), "h", int(minutes), "min", round(seconds), "s beyond 365 days")
gap_min = (SIDEREAL - YEAR) * 24 * 60
print("sidereal minus tropical:", round(gap_min, 1), "minutes")
print("precession cycle:", round(360 * 3600 / 50.29), "years")
print("check:", round(SIDEREAL / (1 + 1 / 25772), 4))
# 5 h 48 min 46 s beyond 365 days
# sidereal minus tropical: 20.4 minutes
# precession cycle: 25771 years
# check: 365.2422
```

So a year is 365 days 5 hours 48 minutes and 46 seconds. There is a second kind of year, too. The **sidereal year** is one orbit measured against the stars, about 365.2564 days, which is 20.4 minutes longer. The difference comes from *precession*: the Earth's axis wobbles slowly like a spinning top, once in about 25,800 years (360 degrees in 360 x 3600 arc-seconds, at about 50.3 arc-seconds a year gives 25,771 years). The equinox point creeps backwards along the Sun's path, so the Sun gets back to the equinox slightly before it gets back to the same stars. Calendars want the seasons to stay put, so they follow the tropical year, not the sidereal year. Had a calendar used the sidereal year, the seasons would move through it by one day every 70 years or so.

Because the figure is a messy fraction, every solar calendar has to carry the extra 0.2422 of a day somewhere. The 19-year trick above is the lunisolar way of hiding the fraction. For a solar calendar the fix is a leap day inserted from time to time, which is the subject of Time: Clocks, Calendars and How We Measure It 4.

> Common mistakes
> Treating "lunar" and "lunisolar" as the same thing (the Islamic calendar has no leap month and drifts through the seasons; the Hebrew and Chinese calendars do not); assuming the Egyptian civil calendar

had leap years (it did not, so its New Year drifted); forgetting that 19 years is 7 leap years of 13 months and 12 ordinary ones, not 19 years of 12 months; confusing the tropical year (the seasons, 365.2422 days) with the sidereal year (the stars, 365.2564); and quoting the 1460-year Egyptian cycle without noting that it assumes a year of exactly 365.25 days.

EXERCISE 1

The Egyptian civil year has 12 x 30 + 5 = 365 days. Find how far its New Year slips against the seasons each year, how many years it takes to slip one season (a quarter of a year), and how many to slip all the way round, using 365.2422 and then exactly 365.25. Compare each result with a purely lunar year of 12 months.

View solution

EXERCISE 2

Compare two ways of choosing the 7 leap years in each 19: the fixed pattern 3, 6, 8, 11, 14, 17, 19 and the "add a month when twelve months would end more than half a month early" rule. For each, print the leap years and the range of the lunar-minus-solar offset over one cycle. Then repeat for 1000 years and report the worst offset reached.

View solution

EXERCISE 3

Build the 30-year tabular Islamic calendar (leap years 2, 5, 7, 10, 13, 16, 18, 21, 24, 26 and 29 have 355 days, the rest 354). Find the days in a cycle, the average month and how many years pass before it is a day out from the true month of 29.530589 days. Then estimate the Gregorian year in which 1, 100, 1000 and 1447 AH began.

View solution

QUICK REFERENCE

- **Lunar calendar:** 12 months of 29 or 30 days, 354 or 355 per year; slips about 10.9 days a year against the seasons, a full lap in about 33.6 years

- **Islamic calendar:** purely lunar, counted from 622 CE; 30-year table with 11 leap years of 355 days = 10631 days
- **Egyptian civil year:** 12 x 30 + 5 = 365 days, no leap days; New Year slips 1 day per 4 years, a full lap in about 1508 years (1460 if the year is taken as 365.25)
- **Lunisolar:** 12 or 13 months; 19 years = 235 months = 228 + 7, matching to 0.087 days; Babylonian, Hebrew and Chinese calendars
- **Hebrew leap years:** years 3, 6, 8, 11, 14, 17 and 19 of each 19-year cycle
- **Tropical year:** 365.2422 days = 365 d 5 h 48 min 46 s; the sidereal year (365.2564 d) is 20.4 minutes longer because of precession

COMING NEXT

Time: Clocks, Calendars and How We Measure It 4 is about the calendar most of the world uses: the Roman muddle, the Julian leap-year reform, the drift that built up, and the Gregorian correction of 1582 that finally brought the leap-year rule to its modern form.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 4 · Julian, Gregorian and the lost days

On Thursday 4 October 1582 the people of Rome went to bed. When they woke, it was Friday 15 October. Nobody slept for ten days: the calendar simply jumped, and the labels 5 to 14 October never existed in Rome that year. Britain did the same trick 170 years later, on 2 September 1752, and woke up on the 14th. Why would a country delete days on purpose? Because Time: Clocks, Calendars and How We Measure It 3 ended with a problem: a year is 365.2422 days, a messy fraction, and every solar calendar has to carry the extra 0.2422 of a day somewhere. This chapter follows the calendar most of the world uses today, from a muddle in ancient Rome to a repair job in 1582, and then through the slow, stubborn way countries adopted it.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The day-number checks compare the Julian rule against Python's own (Gregorian) calendar, so they confirm the 10-day and 11-day gaps independently. The historical details (the pontiffs' extra month, Caesar's 445-day year, Sosigenes, the correction under Augustus, the 1582 bull, the dates different countries switched) are the standard accounts and were written from memory, not looked up again while writing. Treat the ancient dates as "around then" and check any single date before relying on it. The exact length of the tropical year changes very slightly over the centuries; this chapter uses 365.2422 throughout.

The Roman Muddle

The early Roman calendar was lunar in spirit: months followed the Moon, and a normal year had 355 days. That is 10.2422 days short of the solar year (365.2422 - 355), so left alone the seasons would race ahead of the calendar by about a third of a month every year. To pull it back, the priests in charge of the calendar, the *pontiffs*, inserted an extra month, Mercedonius, after 23 February in some years. The textbook pattern was 22 days in one year and 23 days in another, on alternate years. Let us see what that pattern would give if it were followed perfectly, and how a very bad year was finally repaired.

```
year = 355
cycle = [355, 355 + 22, 355, 355 + 23]
```

```
average = sum(cycle) / 4
print("four Roman years:", sum(cycle), "days, average", average)
print("too long by", round(average - 365.2422, 4), "days a year")
print("Year of Confusion, 46 BC:", 355 + 23 + 67, "days")
# four Roman years: 1465 days, average 366.25
# too long by 1.0078 days a year
# Year of Confusion, 46 BC: 445 days
```

Four years of 355 days plus the two extra months come to 4 x 355 + 22 + 23 = 1465 days, an average of 366.25 days a year. That is a whole day too long every year, so even a perfectly followed pattern is not a good one. And it was not followed perfectly. The pontiffs decided when to insert the extra month, and a pontiff could add one to keep a friendly official in office a little longer, or leave it out to cut short the term of a rival. By the middle of the first century BC the calendar was around three months out of step with the seasons: a harvest festival could be marked on the calendar in a month when the grain was still green.

Caesar's Fix: the Julian Calendar

In 46 BC Julius Caesar, with the help of an Alexandrian astronomer usually named as Sosigenes, cut the knot. Two steps:

1. **Reset the clock.** The year 46 BC was stretched to 445 days (the ordinary 355, the usual extra month of 23 days, and two more extra months adding 67 days) to pull the calendar back into line with the seasons. It is remembered as the "year of confusion". 355 + 23 + 67 = 445.
2. **Fix the length.** From 45 BC the year was 365 days, with a leap day added every fourth year. That makes an average of (3 x 365 + 366) / 4 = 1461 / 4 = 365.25 days. The Egyptian civil year of exactly 365 days (Time: Clocks, Calendars and How We Measure It 3) had no leap day and drifted; the Julian calendar adds the missing quarter-day.

The leap day was not put at the end of February. The Romans counted days backwards from fixed dates, and the extra day was made by repeating the day called "the sixth day before the Kalends of March", which is why a leap year was called a *bissextile* year ("twice sixth"). The effect is the same as adding 29 February. The calendar was named the **Julian calendar** after Caesar. An early hiccup is usually reported: the priests counted "every fourth year" in a way that put leap days every third year at first, and the emperor Augustus corrected this by leaving out a few leap years until the pattern settled (traditionally around AD 8).

	ROMAN (BEFORE 46 BC)	JULIAN	GREGORIAN
Rule	355 days plus an extra month the pontiffs decide	a leap day every 4th year	every 4th year, except centuries not divisible by 400
Average year	about 366.25 days if the pattern is followed	365.25 days	365.2425 days
Too long by	about 1 day per year	0.0078 day per year	0.0003 day per year
One day out after	about a year	about 128 years	about 3333 years

The Slow Drift

The Julian year of 365.25 days is a good approximation to 365.2422, but it is a little too long. The error is 365.25 - 365.2422 = 0.0078 of a day each year, which is 0.0078 x 24 x 60 = 11.2 minutes. Tiny, but it never stops adding up. Run the numbers:

```
TROPICAL = 365.2422
calendars = {"Julian": 365.25, "Gregorian": 365.2425}
for name, length in calendars.items():
    error = length - TROPICAL
    print(name, "| error per year", round(error, 4), "d | one day out every", round(1 / 
print("Julian slip, 325 to 1582:", round((1582 - 325) * (365.25 - TROPICAL), 1), "days")
print("one day per", round(1 / (365.25 - TROPICAL), 1), "years")
# Julian | error per year 0.0078 d | one day out every 128 years
# Gregorian | error per year 0.0003 d | one day out every 3333 years
# Julian slip, 325 to 1582: 9.8 days
# one day per 128.2 years
```

One day of error every 128 years: the Julian calendar's date for the equinox slides one day earlier every century and a bit. That mattered to the Church because Easter is defined by the spring equinox. The Council of Nicaea in AD 325 is traditionally said to have settled the rules with the equinox on 21 March. From 325 to 1582 is 1257 years, and 1257 x 0.0078 = 9.8 days: by 1582 the real equinox was falling around 11 March on the Julian calendar. Left alone, Easter would

keep creeping earlier in the calendar year, away from the season it was meant to celebrate. People had noticed for centuries and several scholars had proposed repairs. Eventually the Church, which had the authority and the motive, acted.

The Gregorian Fix of 1582

In February 1582 Pope Gregory XIII issued a decree (the bull *Inter gravissimas*), built on the work of the astronomer Aloysius Lilius and the Jesuit mathematician Christopher Clavius. It did two things:

1. **Drop 10 days** once, so the equinox returned to 21 March, where it had been at Nicaea. In October 1582, 4 October was followed directly by 15 October.
2. **Change the leap-year rule** so the drift would not start again. Century years such as 1700 and 1800 stop being leap years, unless the century is divisible by 400.

```
Rome, October 1582: the days run on, only the labels jump

  Wed 3 Oct  ->  Thu 4 Oct  ->  Fri 15 Oct  ->  Sat 16 Oct
                                ^
                labels 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 never used
```

The new rule in full: a year is a leap year if it is divisible by 4, *except* that century years are leap years only if they are divisible by 400. So 1600 and 2000 were leap years; 1700, 1800 and 1900 were not; 2100 will not be, but 2400 will. Over 400 years the rule leaves out 3 of the 100 leap years the Julian rule would have: 400 x 365 + 97 = 146097 days, an average of 146097 / 400 = 365.2425 days. Here is the whole comparison in code.

```
def leap_julian(y):
    return y % 4 == 0

def leap_gregorian(y):
    return y % 4 == 0 and (y % 100 != 0 or y % 400 == 0)

for name, rule in [("Julian", leap_julian), ("Gregorian", leap_gregorian)]:
    leaps = sum(rule(y) for y in range(2001, 2401))
```

```
    days = 400 * 365 + leaps
    print(name, "| leap years in 400:", leaps, "| days:", days, "| average year:", days /

for y in (1600, 1700, 1800, 1900, 2000, 2100, 2400):
    print(y, "Julian leap?", leap_julian(y), "| Gregorian leap?", leap_gregorian(y))
# Julian | leap years in 400: 100 | days: 146100 | average year: 365.25
# Gregorian | leap years in 400: 97 | days: 146097 | average year: 365.2425
# 1600 Julian leap? True | Gregorian leap? True
# 1700 Julian leap? True | Gregorian leap? False
# 1800 Julian leap? True | Gregorian leap? False
# 1900 Julian leap? True | Gregorian leap? False
# 2000 Julian leap? True | Gregorian leap? True
# 2100 Julian leap? True | Gregorian leap? False
# 2400 Julian leap? True | Gregorian leap? True
```

The error left is 365.2425 - 365.2422 = 0.0003 of a day a year, which takes about 1 / 0.0003 = 3333 years to add up to a day. (The true year itself changes a little over thousands of years, so the figure is only a rough guide.) For everyday purposes the Gregorian calendar is simply correct. Keep one thing from the table of years in mind for the rest of the chapter: 1700, 1800 and 1900 were Julian leap years but not Gregorian ones.

The Calendar That Was Switched Often

A pope could order Catholic countries to change. Everyone else had to decide for themselves, and for many Protestant and Orthodox countries, taking a calendar from the pope looked like a poor idea. The result was a patchwork lasting more than three centuries: for a long time two different dates could be written on the same day, and letters often carried both ("Old Style" for Julian and "New Style" for Gregorian). The dates below are approximate, as usually quoted.

WHERE	SWITCHED	DAYS DROPPED
Italian states, Spain, Portugal	October 1582	10
France	December 1582	10
Protestant German states, Denmark, most of the Netherlands	around 1700	11

WHERE	SWITCHED	DAYS DROPPED
Britain and its colonies	September 1752	11
Sweden	1753	11
Russia	February 1918	13
Greece	1923	13
Japan, China	1873 and 1912	not applicable: they changed from a different calendar

The "days dropped" column grows with time: 10, then 11, then 13. Why? Each time the Gregorian calendar skips a century leap day that the Julian calendar keeps, the gap widens by one day. The Julian calendar kept 1700, 1800 and 1900 as leap years and the Gregorian calendar did not. The gap between the two (Gregorian date minus Julian date) on any date from 1 March of a given year is:

gap = (year // 100) - (year // 400) - 2, where // means divide and drop the remainder.

Try 1582: 15 - 3 - 2 = 10. For 1752: 17 - 4 - 2 = 11. For 1900: 19 - 4 - 2 = 13. The jump happens on 1 March of 1700, 1800 and 1900, because the missing leap day would have been 29 February. The next widening comes on 1 March 2100, taking the gap to 14. Python can check the formula against exact day counts:

```
from datetime import date

def jdn_julian(y, m, d):
    a = (14 - m) // 12
    y2, m2 = y + 4800 - a, m + 12 * a - 3
    return d + (153 * m2 + 2) // 5 + 365 * y2 + y2 // 4 - 32083

def gap(year):
    # Gregorian date minus Julian date, for dates from 1 March of 'year'
    return year // 100 - year // 400 - 2

for y in (325, 1582, 1752, 1900, 2000, 2100):
    print(y, "gap", gap(y), "days")

# check against Python's own (Gregorian) calendar
```

```
    j = jdn_julian(1582, 10, 4)
    g = date(1582, 10, 14).toordinal() + 1721425
    print("Julian 4 Oct 1582 is Gregorian 14 Oct:", j == g)
    print("Julian 3 Sep 1752 is Gregorian 14 Sep:", jdn_julian(1752, 9, 3) == date(1752, 9,
    print(date(1582, 10, 15).strftime("%A"), date(1752, 9, 14).strftime("%A"))
    # 325 gap 1 days
    # 1582 gap 10 days
    # 1752 gap 11 days
    # 1900 gap 13 days
    # 2000 gap 13 days
    # 2100 gap 14 days
    # Julian 4 Oct 1582 is Gregorian 14 Oct: True
    # Julian 3 Sep 1752 is Gregorian 14 Sep: True
    # Friday Thursday
```

The checks agree: Julian 4 October 1582 is the Gregorian 14 October, and the next day, Julian 5 October, is the Gregorian 15 October (and so Thursday was followed by Friday: only the labels jumped, never the weekdays). The same check for Britain shows that Julian 3 September 1752 is Gregorian 14 September, so the dropped labels there were 3 to 13 September, 11 of them.

Britain's Short Years

The British change came with a second shift, which is why the story is a little more complicated than "11 days". The Calendar (New Style) Act of 1750 did two things. It dropped 11 days in September 1752, and it moved the start of the legal year from 25 March (Lady Day) to 1 January. Because the new rule began with the year 1752, the year 1751 was shortened:

```
    from datetime import date
    short_1751 = (date(1751, 12, 31) - date(1751, 3, 25)).days + 1
    print("British year 1751 (25 March to 31 December):", short_1751, "days")
    print("British year 1752:", 366 - 11, "days (leap year, 11 days dropped)")
    # British year 1751 (25 March to 31 December): 282 days
    # British year 1752: 355 days (leap year, 11 days dropped)
```

The year 1751 ran only from 25 March to 31 December, 282 days (7 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 + 31 = 282), and 1752 lost 11 days from a leap year's 366, leaving 355. A story says that crowds rioted with the cry "Give us our eleven days", believing the government had stolen 11 days of their lives. Historians have found little evidence for the riots, and the tale seems to owe much to a satirical painting. Real concerns were more practical: rents, wages and interest had to be adjusted, and people argued about whether a quarter's rent was due on the old date or the new one.

The leftover mess shows up in historical records. George Washington's birthday is given as 11 February 1731 in the old British system and 22 February 1732 in the new one. The Russian "October Revolution" took place on 25 October 1917 in the calendar Russia was using, which is 7 November in the Gregorian calendar, a difference of 13 days. Some Orthodox churches still keep Julian dates for their feasts, which is why their Christmas lands on 7 January: 25 December plus 13 days of gap. Exercise 3 works these conversions out.

Common mistakes

Saying the Gregorian reform "dropped 10 days" everywhere (it dropped 10 in 1582 but 11 in Britain in 1752 and 13 in Russia in 1918, because the gap widens with each skipped century leap day); assuming days of the week were skipped (they were not: only the date labels jumped); believing every year divisible by 100 is a leap year (1900 was not, 2000 was); treating 2100 as a leap year; forgetting that the British year used to start on 25 March when reading old dates; and assuming the whole world changed in 1582 (many places waited centuries).

EXERCISE 1

Write two functions, one for the Julian leap-year rule and one for the Gregorian rule. List the years between 1580 and 1720 where they disagree. Then count the leap years in 400 consecutive years under each rule and report the average year length, the yearly error against 365.2422 and how many years it takes to build up one day of error.

View solution

EXERCISE 2

Use the formula gap = year // 100 - year // 400 - 2 to print the gap between the calendars for 325, 1582, 1700, 1752, 1900, 2000 and 2100. Check it against exact day numbers for a handful of dates, and confirm that Julian 5 October 1582 is the Gregorian 15 October and that Julian 3 September 1752 is the Gregorian 14 September.

View solution

EXERCISE 3

Convert these Julian dates to Gregorian: Washington's birthday (11 February 1732 counted from 1 January), the Russian "October" Revolution (25 October 1917) and 25 December 2025. Then work out how many days the British year 1751 had, running from 25 March to 31 December, and how many days 1752 had.

View solution

QUICK REFERENCE

- **Roman calendar:** 355-day year plus an extra month decided by the pontiffs; by the 40s BC it was about three months adrift
- **Julian calendar (45 BC):** 365 days plus a leap day every 4th year = 365.25 days; too long by 0.0078 day a year, one day out every 128 years
- **Drift by 1582:** 1257 years since Nicaea x 0.0078 = about 9.8 days, so the equinox fell around 11 March
- **Gregorian rule (1582):** leap if divisible by 4, except centuries not divisible by 400; 97 leap years in 400 = 365.2425 days; one day out every 3333 years or so
- **Gap, Gregorian minus Julian:** year // 100 - year // 400 - 2 (from 1 March): 10 in 1582, 11 in 1752, 13 from 1900, 14 from 1 March 2100
- **Switch dates (approximate):** Catholic Europe 1582, Britain 1752 (2 September to 14 September), Russia 1918, Greece 1923
- **Britain 1751 and 1752:** 1751 had only 282 days (25 March to 31 December); 1752 had 355

COMING NEXT

Time: Clocks, Calendars and How We Measure It 5 looks at the pieces a calendar is built from: why there are seven days in a week, where the months got their names, and how years have been numbered, from the founding of Rome to the reigns of kings and emperors.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 5 · Weeks, months and eras

Look at a calendar and ask what in it comes from the sky. The day comes from the Earth's spin, the month from the Moon, the year from the Sun. The week comes from nothing in the sky at all: seven days does not divide into a month or a year, and no heavenly body repeats every seven days. Yet it is the most stubborn rhythm we have. When the Julian calendar gave way to the Gregorian in Time: Clocks, Calendars and How We Measure It 4, whole date labels vanished, but the weekday never wavered: Thursday 4 October 1582 was followed by Friday 15 October. This chapter looks at the pieces a calendar is stitched together from: weeks, months and the numbers we hang on years.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The arithmetic (day counts, week counts, year conversions) is checked by that code. The historical origins of the week and of month names are partly traditional and partly debated; where scholars disagree, the text says so.

Where the Seven-Day Week Came From

Nobody knows for certain, but two threads are clear. The first is the Moon. A lunar month is 29.53 days, and a quarter of it is 29.53 / 4 = 7.38 days, so the four phases (new, first quarter, full, last quarter) fall about seven days apart. Babylonian records treat certain days of the lunar month, around the 7th, 14th, 21st and 28th, as special. The second thread is Jewish: well before the Christian era a seven-day cycle ending in a Sabbath was running as a pure counting cycle, rolling on week after week whatever the Moon did. A week that does not restart with each month is the real invention.

The names came from a third source. Greek-speaking astronomers of the Hellenistic world knew seven wandering bodies: Saturn, Jupiter, Mars, the Sun, Venus, Mercury and the Moon. Ordered from slowest-moving to fastest-moving against the stars, that is the list just given. By around the first century AD a planetary week was in use in the Roman world, with the days named after those seven. The way the order was chosen is a neat puzzle, and the arithmetic is simple:

1. Give each of the 24 hours of a day to one planet in turn, following the list and wrapping round: Saturn, Jupiter, Mars, Sun, Venus, Mercury, Moon, Saturn again, and so on.
2. Name each day after the planet that rules its *first* hour.
3. Because 24 = 3 x 7 + 3, after 24 hours the list has gone round three full times and then three places further. So the next day starts three places along the list.

```
planets = ["Saturn", "Jupiter", "Mars", "Sun", "Venus", "Mercury", "Moon"]
print("24 mod 7 =", 24 % 7)
day_rulers = [planets[(24 * day) % 7] for day in range(7)]
print(day_rulers)
# 24 mod 7 = 3
# ['Saturn', 'Sun', 'Moon', 'Mars', 'Mercury', 'Jupiter', 'Venus']
```

Count three places along from Saturn: Jupiter, Mars, **Sun**. Three more from the Sun: Venus, Mercury, **Moon**. Three more from the Moon (wrapping round to the start): Saturn, Jupiter, **Mars**. The sequence Saturn, Sun, Moon, Mars, Mercury, Jupiter, Venus is exactly the order of the days from Saturday to Friday. Because 3 and 7 share no common factor, a step of three visits all seven planets before repeating, which is why every planet gets exactly one day.

The names survive in many languages. English Saturday, Sunday and Monday are Saturn's day, the Sun's day and the Moon's day directly. For Tuesday to Friday, English used Germanic gods in place of the Roman ones: Tiw (matching Mars), Woden (Mercury), Thor (Jupiter) and Frigg (Venus). French keeps the Roman gods: *mardi* (Mars), *mercredi* (Mercury), *jeudi* (Jupiter), *vendredi* (Venus).

The week was fixed in public life by religion and law. Judaism, Christianity and Islam each gave one day of the seven a special status (Saturday, Sunday and Friday). In AD 321 the emperor Constantine made Sunday a day of rest in the cities of the Roman Empire. A cycle backed by religion spread with the faiths and never lost its place. This is also why the calendar changes in Time: Clocks, Calendars and How We Measure It 4 left the weekday alone: weekdays had been counted without a break from long before 1582, and nobody wanted to disturb the rhythm of a day of worship.

Months and Their Names

The months are the oldest unit in this chapter, but the names we use are Roman and carry fossils of an earlier calendar. According to Roman tradition the original year had ten months, beginning in March and ending in December, followed by a stretch of winter days with no month name at all. The tradition is old but not certain: ancient writers disagreed about details. What is plain is the evidence in the names themselves. September, October, November and December mean 7th, 8th, 9th and 10th month (from the Latin *septem*, *octo*, *novem*, *decem*), which only makes sense if March is month 1.

```
months = ["March", "April", "May", "June", "Quintilis", "Sextilis",
          "September", "October", "November", "December"]
for number in (5, 6, 7, 8, 9, 10):
    print(number, months[number - 1])
print("ten months, 304 days; 365 - 304 =", 365 - 304, "days left over (the winter gap)")
print("31-day months:", 7, "| 30-day:", 4, "| February: 28 | check:", 7 * 31 + 4 * 30 + 
# 5 Quintilis
# 6 Sextilis
# 7 September
# 8 October
# 9 November
# 10 December
# ten months, 304 days; 365 - 304 = 61 days left over (the winter gap)
# 31-day months: 7 | 30-day: 4 | February: 28 | check: 365
```

A ten-month year of 304 days (the commonly reported total) leaves 365 - 304 = 61 days spare, which fits the story of an unnamed winter gap. January and February were later added, and eventually placed at the front of the year, so that the "7th" month became the 9th, the "8th" the 10th and so on. That is why the names have been two numbers wrong ever since.

MONTH	NAMED FOR	NOTE
January	Janus, god of doorways and beginnings	a fitting name for the first month once the year start moved
February	*februa*, Roman purification rites	the short winter month where leap days live
March	Mars, god of war	the original first month

MONTH	NAMED FOR	NOTE
April, May, June	origins debated	May and June may honour Maia and Juno, but ancient writers did not agree
July	Julius Caesar	renamed from Quintilis (the "fifth") in 44 BC, after his death
August	Augustus	renamed from Sextilis (the "sixth") in 8 BC
September to December	the numbers 7 to 10	two places out of step since January and February were added

You may have heard that August has 31 days because Augustus wanted to be as important as Julius Caesar and took a day from February. It makes a good story, but historians doubt it: the Julian calendar's month lengths seem to have been set in 45 BC, before Augustus. What is certain is the sum. Seven months of 31 days, four of 30 and one of 28 give 7 x 31 + 4 x 30 + 28 = 217 + 120 + 28 = 365, as the code above confirms.

Numbering the Years

A calendar tells you the day within a year; an *era* tells you which year. Every culture picked a starting point that mattered to it. Four systems cover most of the history you will meet:

SYSTEM	COUNTS FROM	EXAMPLE
AUC (*ab urbe condita*)	the traditional founding of Rome, 753 BC	AD 1 = 754 AUC; 2026 = 2779 AUC
AD / BC (and CE / BCE)	a calculated date for the birth of Jesus	CE and BCE are the same numbers with neutral names
Regnal years	the accession of the current ruler	"in the 3rd year of Queen Anne"
Japanese eras (*nengo*)	a new era name for each reign	Reiwa 1 = 2019

The Romans themselves often named a year by its two consuls ("in the consulship of...") and only sometimes counted from the founding of the city, which the scholar Varro placed in 753 BC. The AD system is later: in AD 525 a monk named Dionysius Exiguus, preparing Easter tables, proposed numbering years from the incarnation of Jesus, placed so that AD 1 came 754 years

after the founding of Rome. The system was taken up gradually, helped by the English monk Bede, and became standard in Europe over the following centuries. Historians today generally think Jesus was born a few years earlier than Dionysius's calculation implies, but the labels were never corrected, since renumbering would only cause confusion.

There is one famous trap. Dionysius had no number zero (Roman numerals did not have one), so the year before AD 1 is 1 BC. There is no year 0, which has a side effect: the span from 1 BC to AD 1 is a single year. Astronomers, who need their arithmetic to work smoothly, number the years with a year 0. In that scheme 1 BC is 0, 2 BC is -1, and in general **astronomical year = 1 - BC year**. So the "year of confusion" of 46 BC in Time: Clocks, Calendars and How We Measure It 4 is -45 to an astronomer.

```
def ad_to_auc(year):
    return year + 753

def bc_to_astronomical(bc):
    return 1 - bc

print("AD 525 (Dionysius's own time):", ad_to_auc(525), "AUC")
print("AD 1 =", ad_to_auc(1), "AUC")
print("2026 =", ad_to_auc(2026), "AUC")
print("1 BC -> astronomical year", bc_to_astronomical(1))
print("45 BC -> astronomical year", bc_to_astronomical(45))
# AD 525 (Dionysius's own time): 1278 AUC
# AD 1 = 754 AUC
# 2026 = 2779 AUC
# 1 BC -> astronomical year 0
# 45 BC -> astronomical year -44
```

The rule for AUC is "add 753" for an AD year, because AD 1 = 754. For a BC year, subtract it from 754: 753 BC = 754 - 753 = 1 AUC.

Counting by reigns is the oldest and most natural method: you know who is king, so you count his years. English statutes were dated that way until 1962. Japan still does it. A new era name (*nengo*) is chosen when an emperor takes the throne: Showa began in 1926, Heisei in 1989 and Reiwa on 1 May 2019. To convert a Gregorian year to an era year, subtract the era's first year and add one, since the first year of an era is year 1: 2026 - 2019 + 1 = Reiwa 8. In the year of a

change two eras share the year, since the switch happens partway through (Heisei 31 and Reiwa 1 are both parts of 2019). The months and days stay Gregorian; only the year label is the era.

Counting Weeks: ISO 8601

Businesses and logistics often need to say "week 46" and mean the same week everywhere. The international standard ISO 8601 gives the rules:

1. Weeks run from Monday (day 1) to Sunday (day 7).
2. Week 1 of a year is the week that contains that year's **first Thursday**. Equivalently, it is the week containing 4 January.
3. A date is written `year-Wweek-day`, for example `2026-W46-1`.

Why Thursday? A Monday-to-Sunday week has its Thursday in the middle, and whichever year holds the Thursday holds at least four of the week's seven days. So each week belongs wholly to one year. A side effect is that the first days of January can belong to week 52 or 53 of the old year, and the last days of December can belong to week 1 of the next:

```
from datetime import date
for d in (date(2026, 11, 9), date(2026, 12, 31), date(2027, 1, 1), date(2027, 1, 4), dat
    year, week, day = d.isocalendar()
    print(d, "->", f"{year}-W{week:02d}-{day}", d.strftime("%A"))
# 2026-11-09 -> 2026-W46-1 Monday
# 2026-12-31 -> 2026-W53-4 Thursday
# 2027-01-01 -> 2026-W53-5 Friday
# 2027-01-04 -> 2027-W01-1 Monday
# 2025-12-29 -> 2026-W01-1 Monday
```

1 January 2027 is a Friday, and its week holds Thursday 31 December 2026, so it belongs to week 53 of 2026. Meanwhile Monday 29 December 2025 already counts as week 1 of 2026. A year has 52 or 53 ISO weeks: 53 when 1 January is a Thursday, or in a leap year when it is a Wednesday. Over the 400-year Gregorian cycle the totals are neat: 146097 days is exactly 20871 weeks (146097 / 7 = 20871), so the Gregorian calendar repeats its weekdays every 400 years. Of those 400 years, 71 have 53 weeks (52 x 400 + 71 = 20871).

Other Weeks That Were Tried

The seven-day week is a convention, and people have tried others.

WEEK	WHERE AND WHEN	WHAT HAPPENED
8 days	Roman market cycle (*nundinae*)	a market day every eighth day (called the ninth by Roman inclusive counting); faded as the seven-day week spread
10 days	French Republican calendar, from 1793	each month was three ten-day *decades*, with one rest day in ten; abolished by Napoleon from 1 January 1806
5 days	Soviet Union, from 1929	a continuous work week to keep factories running; disliked and replaced
6 days	Soviet Union, from 1931	replaced the 5-day week; the seven-day week came back in 1940

The French ten-day week gave workers one day off in ten instead of one in seven, and it was never popular. The Soviet experiments caused trouble for a different reason: with staggered rest days, families and friends often had different days off, so ordinary social life suffered. A week works because everyone shares it, and the seven-day cycle won largely because the surrounding world already used it.

> Common mistakes
> Believing the week comes from the Moon or the Sun (it is a counting cycle, with a loose link to the Moon's quarters); forgetting that there is no year 0 between 1 BC and AD 1; mixing up the astronomical year (1 BC = 0) with the BC label; assuming ISO week 1 always starts on 1 January (it is the week holding the first Thursday); and assuming every year has 52 weeks (some have 53).

EXERCISE 1

Write a short program that assigns the 24 hours of a day to the planets in the ancient order (Saturn, Jupiter, Mars, Sun, Venus, Mercury, Moon) and names each of seven consecutive days after the planet ruling its first hour, starting from Saturn's day. Show that 24 mod 7 = 3 explains the pattern, and check your result against the real order of the weekdays.

View solution

EXERCISE 2

Write functions to convert an AD year to AUC, a BC year to AUC, a BC year to astronomical numbering, and a Gregorian year to a Japanese era year (Meiji 1868, Taisho 1912, Showa 1926, Heisei 1989, Reiwa 2019). Try 2026 AD, 753 BC, 44 BC, 1926, 1989 and 2019.

View solution

EXERCISE 3

Write your own function that returns the ISO week year and week number of a date using the first-Thursday rule, and compare it with Python's `isocalendar()` for every day from 1900 to 2100. Then list the 53-week years from 2015 to 2040, and show that 400 Gregorian years contain exactly 20871 weeks, with 71 of those years having 53 weeks.

View solution

QUICK REFERENCE

- **Week:** a pure counting cycle; a quarter of the lunar month is 29.53 / 4 = 7.38 days, so the Moon is a loose cousin, not the cause
- **Planetary week:** 24 mod 7 = 3, so each day starts three places on in the list Saturn, Jupiter, Mars, Sun, Venus, Mercury, Moon, giving Saturn, Sun, Moon, Mars, Mercury, Jupiter, Venus
- **Months:** September to December are the 7th to 10th months of the early March-start year; 7 x 31 + 4 x 30 + 28 = 365
- **AUC:** AD year + 753 (AD 1 = 754 AUC); BC year: 754 - BC
- **Astronomical year:** 1 - BC year (1 BC = 0); there is no year 0 on the BC/AD scale
- **Japanese era year:** Gregorian year - era start year + 1 (2026 = Reiwa 8)
- **ISO 8601:** Monday start, week 1 holds the first Thursday (contains 4 January); 52 or 53 weeks a year; 146097 days = 20871 weeks per 400 years, 71 of which are 53-week years
- **Other weeks:** 8-day Roman, 10-day French Republican (1793 to 1806), 5-day and 6-day Soviet (1929 to 1940)

COMING NEXT

Time: Clocks, Calendars and How We Measure It 6 moves from the calendar to the machine: the first mechanical clocks of medieval Europe, why bells came before faces, and the escapement that turned a falling weight into a steady count.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 6 · The mechanical clock

Here is an odd fact about the word "clock": it comes from the word for a *bell*. Medieval Latin *clocca*, French *cloche* and Dutch *klok* all mean bell, and the English word was borrowed from that family. The earliest mechanical clocks of Europe were not made to be looked at. They were made to be *heard*, and many had no dial at all. To see why, we need to start where Time: Clocks, Calendars and How We Measure It 2 left off, with water clocks and sundials, and ask what job a clock was first built to do.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The arithmetic (error fractions, energy, power, beat counts) is checked by that code. The early history of mechanical clocks is thinly documented: nobody knows who invented the escapement or exactly when, and the figures for how well early clocks kept time are rough, so the text says "about" where scholars only have estimates.

Monks, Bells and the Hours

The pressure for a better clock came partly from monasteries. The Rule of St Benedict, written around AD 530, set out a day built round prayer, and over the centuries this grew into the **canonical hours**: matins in the night, then lauds, prime, terce, sext, none, vespers and compline. The names of the daytime offices are just numbers. Prime is the "first" hour, terce the "third", sext the "sixth" and none the "ninth", counted in the seasonal hours from Time: Clocks, Calendars and How We Measure It 2, so they shifted with the seasons.

Somebody had to wake the community and ring the bell at the right moment, including in the dark. Water clocks, with an alarm that sounded a bell, were used for this, and so were candles and watchful bell-ringers. A mechanism that struck a bell on its own, without an attendant watching a water level, was just what a monastery or a town church wanted. A bell is also the right tool for the job in a different sense: its sound carries over roofs and fields to everyone who works there, and most people could not read a dial anyway.

	WATER CLOCK	EARLY MECHANICAL CLOCK
Power	water flowing out of a vessel	a heavy weight on a rope
What sets the rate	the steady flow of water	an oscillating bar, the foliot (below)
Needs	refilling the tank	hauling the weight back up
Main weakness	flow changes with temperature and water level	friction and a poorly steady oscillation
Natural output	a level on a scale	a counted tick, easy to turn into bell strikes

Why Bells Came Before Faces

A machine that counts ticks can ring a bell with very little extra gear: after so many ticks, lift a hammer and let it fall. Making a dial is harder. You need a train of gears that turns a hand at exactly the right slow speed, a wheel for the hours and one for the minutes, and a face that can be read from a distance. Clocks that struck bells appeared in the late thirteenth century (a clock at Dunstable Priory in England is recorded in 1283), and the early ones seem to have had no hands. Dials and hour hands arrived later, during the fourteenth century. The phrase "*o'clock*" is a leftover from that age: it is short for "of the clock", meaning the hour that the clock *sounded*.

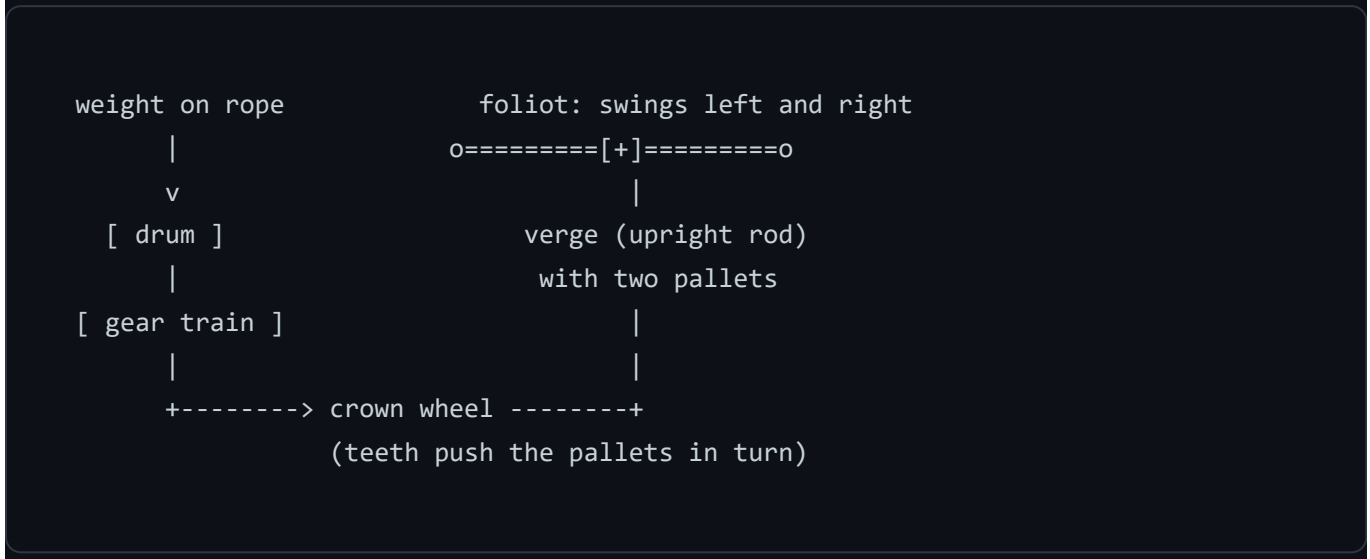
Almost none of the earliest machines survive: iron parts were reused, and clock after clock was replaced by a better one. One of the oldest working examples is the Salisbury Cathedral clock, made around 1386, which has no dial at all and simply strikes the hours on a bell. It was repaired and reassembled over the centuries, so how much of it is original is a matter of study, but it shows what the first clocks were: iron frames the size of a table that announced the hour.

The Escapement: Letting Energy Out in Steps

A falling weight on its own would simply fall. To make it a clock you must make it fall *slowly and evenly* and count how far it has gone. That is the job of the **escapement**, a part which lets the driving wheel advance one small step at a time. The idea needs three pieces:

1. A **drive**: a weight on a rope wound round a drum, turning a train of gears to the **escape wheel** (in early clocks a "crown" wheel, with teeth standing up like the points of a crown).

2. A **verge**: an upright rod with two small flat plates, called **pallets**, set about a right angle apart so that each one catches the wheel on opposite sides.
3. A **foliot**: a horizontal bar fixed to the top of the verge, with a weight at each end, that swings to and fro like a slow see-saw.



Here is how one beat goes. A tooth of the crown wheel pushes on one pallet. The push turns the verge, which swings the foliot round. By then the other pallet has moved into the path of a tooth on the opposite side of the wheel, and it blocks the wheel. The foliot, still moving, carries on until the blocked tooth pushes it back the other way, and the cycle repeats. Each swing from one side to the other is a **beat**, and each beat lets the wheel advance by half a tooth. That is the characteristic tick of a verge clock: tick-tock, tick-tock, one beat each.

The speed of the clock is set by how long the foliot takes to swing, and that depends on how hard it is to turn. A foliot with heavy weights far out along the bar is hard to turn, so it swings slowly; moving the weights inward makes it swing faster and the clock gain. Adjusting the weights was how a clock-keeper set the rate. You can already see the weak point: nothing in a foliot pulls it back to its centre with a steady strength, as gravity does for a pendulum. The only thing setting its swing is the push from the drive, so if friction rises or the weight is changed, the rate wanders. Time: Clocks, Calendars and How We Measure It 7 will show how the pendulum and the balance spring fixed this by giving the oscillator a rhythm of its own.

How Wrong Were They?

Early tower clocks were not precise. Estimates of an error of the order of a quarter of an hour a day are common, so the clock-keeper checked the clock against a sundial or the noon Sun and

reset it, perhaps weekly. Let us put figures on that error with the vocabulary from Time: Clocks, Calendars and How We Measure It 1.

```
SECONDS_PER_DAY = 86400
error = 15 * 60        # a quarter of an hour, in seconds
print("fraction wrong:", round(error / SECONDS_PER_DAY * 100, 2), "percent")
print("about 1 part in", round(SECONDS_PER_DAY / error))
print("after a week:", 15 * 7, "minutes if never reset")
# fraction wrong: 1.04 percent
# about 1 part in 96
# after a week: 105 minutes if never reset
```

A quarter of an hour in 24 hours is 900 / 86400 = 0.0104, about one percent: the clock is right to about 1 part in 96. That sounds poor, but it was good enough for its job. A bell for prayer or a market only needed the hour roughly right, and a clock that gained or lost a quarter of an hour in a day, reset every week or so, still marked the hours better than a tired watchman on a cold night. It was not good enough for astronomy, and it was nowhere near enough for navigation, which is the problem Time: Clocks, Calendars and How We Measure It 8 turns to.

What Powers It

A weight-driven clock runs on very little power. The energy of a falling weight is its mass times *g* (9.81 metres per second squared, the pull of gravity) times the distance it drops. A 100 kg weight falling 10 m gives 100 x 9.81 x 10 = 9810 joules, and spread over a day of 86400 seconds that is a power of 9810 / 86400 = 0.1135 watts.

```
mass, drop, g = 100, 10, 9.81
energy = mass * g * drop
print("energy per day (J):", energy)
print("power (W):", round(energy / 86400, 4))
# energy per day (J): 9810.0
# power (W): 0.1135
```

That is about a tenth of a watt: roughly 1/880 of what a 100 W light bulb draws. The clock-keeper restored the energy by winding the weight back up, doing the same 9810 joules of work by hand. All of that energy goes through the escapement in tiny steps, which is why the escapement, and not the weight, is the heart of the clock.

Common mistakes

Thinking the first clocks had dials (many had only a bell); believing the weight is what keeps time (it only supplies energy, while the escapement and the oscillator set the rate); assuming the foliot behaves like a pendulum (it has no restoring force of its own); and treating the monastic hours as fixed clock times (they were seasonal hours until equal hours took over).

EXERCISE 1

An idealised verge escapement has a crown wheel with 15 teeth, and each beat of the foliot takes 2.0 seconds. Work out how many beats the wheel needs for one full turn, how many seconds one turn takes, and how many beats, teeth released and turns occur in a day. Then suppose the foliot weights drift outward so each beat takes 2.02 seconds while the gear train still counts it as 2.00: how many minutes a day does the clock lose? What beat length would lose a quarter of an hour a day?

View solution

EXERCISE 2

Treat the daylight hours as 12 "day-hours" (Time: Clocks, Calendars and How We Measure It 2). For a winter day with light from 08:40 to 15:40 and a summer day from 04:30 to 21:30, find the length of a day-hour in minutes and the modern clock times of prime (sunrise), terce, sext and none (taken as 3, 6 and 9 day-hours after sunrise). How far does none move between winter and summer? What does that suggest about why a fixed-hour mechanical clock changed how people used these names?

View solution

EXERCISE 3

Write functions for the energy of a falling weight and the power it delivers over a day. Find both for a 100 kg weight dropping 10 m, and compare the power with a 100 W bulb. Then work out how many days a 30 m drop lasts at 10 m a day, and what mass would be needed to deliver 0.5 W from a 10 m daily drop.

View solution

QUICK REFERENCE

- **Clock:** from medieval Latin *clocca*, a bell; the first mechanical clocks struck bells and often had no dial
- **Canonical hours:** matins, lauds, prime, terce, sext, none, vespers, compline; counted in seasonal hours
- **Escapement:** lets the drive wheel advance one step per beat; in a verge clock a crown wheel, a verge with two pallets and a foliot bar
- **Beat:** one swing of the foliot; each beat advances the wheel by half a tooth
- **Rate control:** weights on the foliot; farther out means slower
- **Typical error:** of the order of 15 minutes a day, 900 / 86400 = 1.04 percent, about 1 part in 96
- **Power:** energy = mass x g x drop; 100 kg dropping 10 m a day = 9810 J = 0.1135 W

COMING NEXT

Time: Clocks, Calendars and How We Measure It 7 gives the clock an oscillator with a mind of its own: Galileo's pendulum, Huygens's pendulum clock, the anchor escapement and the balance spring, with the pendulum period formula worked and simulated.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 7 · Pendulums and springs

The verge-and-foliot clock of Time: Clocks, Calendars and How We Measure It 6 had a weakness at its heart: its swinging bar had no rhythm of its own. Push it harder and it swung faster, add friction and it slowed. What clockmakers wanted was an oscillator that *insists* on a particular beat whatever the drive does. They found two: a weight on a string, pulled back by gravity, and a wheel on a spring, pulled back by the spring. Both gave a leap in accuracy within about twenty years of each other, in the 1650s and 1670s.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The formulas are standard small-swing results. Dates and credits in the history are the commonly given ones, but several are disputed or come from later retellings (the Galileo lamp story especially), and the accuracy figures for early clocks are rough estimates, so the text says "about".

Galileo's Lamp

The story goes that the young Galileo, sitting in the cathedral at Pisa around 1583, watched a lamp swing on its chain and timed the swings against his own pulse. He noticed that a wide swing and a narrow swing took about the same time. This tale was written down decades later by his pupil Vincenzo Viviani, so treat the details with care, but the observation itself is real: Galileo did study pendulums and found that the time of a swing depends on the length of the string and hardly at all on the size of the swing or the weight of the bob.

"Hardly at all" is the key phrase, and we will come back to it. First, the formula. For a small swing, the time of one full to-and-fro, the **period** T, is

$$T = 2\pi \times \sqrt{(L / g)}$$

where L is the length of the pendulum in metres and $g = 9.81$ m/s^2 is the pull of gravity. Two things stand out. Weight is missing: a heavy bob and a light one swing together. And the length sits under a square root, so to double the period you must make the pendulum *four* times as long.

```
import math

g = 9.81

def period(length):
    return 2 * math.pi * math.sqrt(length / g)

def length_for(T):
    return g * (T / (2 * math.pi)) ** 2

print("1 m pendulum:", round(period(1.0), 4), "s")
print("length for a 2 s period:", round(length_for(2.0), 4), "m")
print("length for a 1 s period:", round(length_for(1.0), 4), "m")
# 1 m pendulum: 2.0061 s
# length for a 2 s period: 0.994 m
# length for a 1 s period: 0.2485 m
```

A pendulum a little under a metre long (0.994 m) takes 2 seconds for a full swing, one second out and one second back. That is the "seconds pendulum", the length behind the long cases of grandfather clocks: the tall case is there to give room for a rod that long. Check it by hand: 2π = 6.283, 0.994 / 9.81 = 0.1013, whose square root is 0.3183, and 6.283 × 0.3183 = 2.00.

Huygens Builds the Pendulum Clock

Galileo apparently designed a pendulum clock late in life but never finished one. The Dutch scientist **Christiaan Huygens** did. He built the first working pendulum clock at the end of 1656, patented it in 1657, and the clockmaker Salomon Coster of The Hague began making them. In 1673 Huygens published the full theory in a book, *Horologium Oscillatorium*. The improvement was dramatic: where a good verge clock might be off by a quarter of an hour a day, Huygens's early pendulum clocks kept time to within about a minute a day, and refined ones reached roughly ten seconds a day. In terms of Time: Clocks, Calendars and How We Measure It 1, a clock many times better had arrived almost overnight.

The pendulum's rhythm belongs to the pendulum, not to the drive. The weight only needs to give the bob a small push each swing, to replace the energy lost to friction and air. A clock good

to a minute a day was also worth having a minute hand for, and minute hands became common on clocks after this time.

The Circular Error

Remember "hardly at all"? A real pendulum does slow down slightly for bigger swings. For an amplitude of θ radians the period grows by roughly $\theta^2 / 16$, as a fraction. Take a 5 degree swing: 5 degrees = 0.0873 radians, squared is 0.00762, divided by 16 is 0.000476, which is 0.0476 percent. A day has 86400 seconds, so that is 86400 × 0.000476 = 41 seconds a day of extra slowness compared with a vanishingly small swing.

```
import math

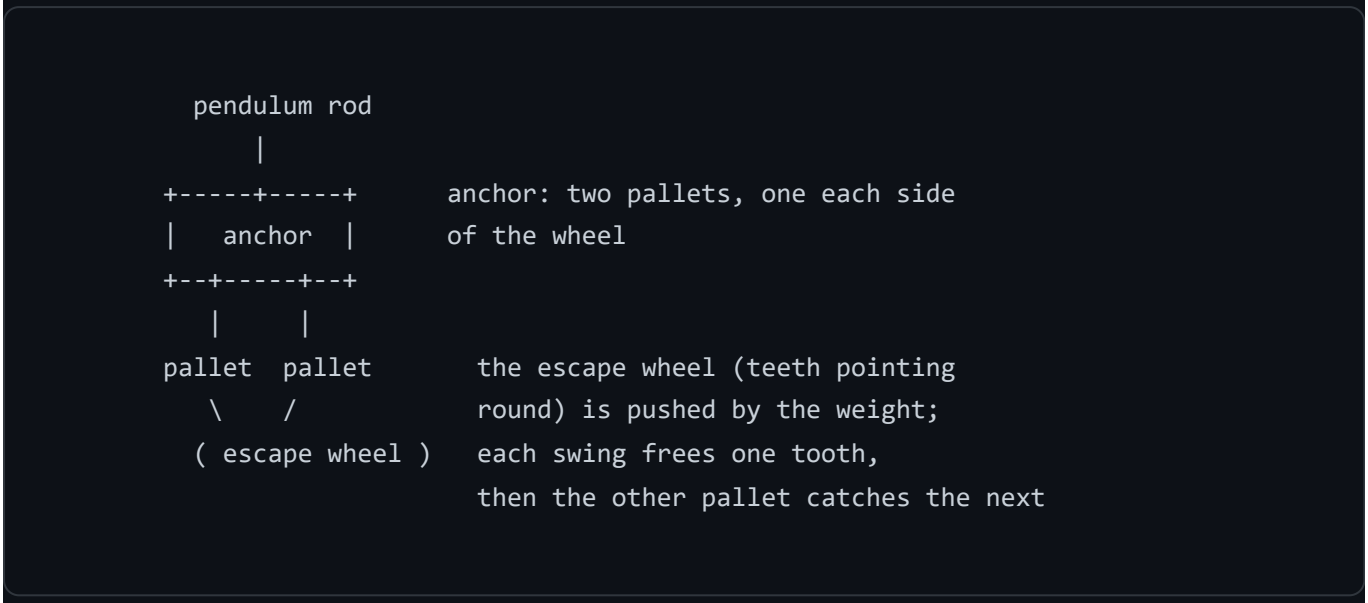
for deg in (2, 5, 10, 30):
    theta = math.radians(deg)
    fraction = theta ** 2 / 16
    print(deg, "deg:", round(fraction * 100, 4), "percent slower,", round(fraction * 864
# 2 deg: 0.0076 percent slower, 6.6 s a day
# 5 deg: 0.0476 percent slower, 41.1 s a day
# 10 deg: 0.1904 percent slower, 164.5 s a day
# 30 deg: 1.7135 percent slower, 1480.4 s a day
```

This **circular error** matters in two ways. First, a clock whose swing changes (because the drive weakens, or the oil thickens) changes its rate by this amount, so small swings are far better than large ones. Second, it shaped the design of the clocks. Huygens tried to cure the error by making the bob follow a special curve (a cycloid) using curved metal cheeks beside the suspension. It worked in theory, but the cheeks added friction, and later makers simply kept the arc small.

The Anchor Escapement

The verge needed a wide swing, far too wide for a good pendulum. What was needed was a new escapement that worked with a small arc. Around 1670 the **anchor escapement** appeared, usually credited to Robert Hooke and the clockmaker William Clement (the credit is debated).

The wheel is now an upright **escape wheel** with pointed teeth, and the part that catches it is shaped like a ship's anchor, hence the name.



One tooth is released at each swing, and as it slides past a pallet it gives the pendulum a tiny push. That push is exactly the energy the pendulum lost. The swing needs only about 4 to 6 degrees, so the circular error is small. In the 1710s and 1720s George Graham refined it into the **dead-beat escapement**, in which the wheel does not recoil: in the plain anchor, the wheel is pushed back a little at each swing, which is what makes a tick look like a shiver in the second hand. A long, slow pendulum then fitted in a tall case, which is how the "longcase" or grandfather clock took shape.

The Balance Spring

A pendulum needs gravity, and gravity needs a clock that stays upright and still. Carry a pendulum on a ship or in a pocket and it fails. The portable answer was a **balance wheel**, a small ring that rotates to and fro, but a plain wheel has no pull-back force, so it behaves like the foliot. Fitting a thin **balance spring** (the hairspring) gave it one. Huygens designed a spiral spring in 1675, and Robert Hooke claimed that he had invented the idea years earlier, which led to a famous dispute. Hooke's law (a spring pulls back in proportion to how far it is stretched) is the physics behind it.

The period of a balance wheel is $T = 2\pi \times \sqrt{(I / k)}$, where I is how hard the wheel is to turn (its moment of inertia) and k is the stiffness of the spring. There is no g in this formula, so the watch works in any position and on a moving ship. Its beat can also be fast: a common modern watch beats 18000 times an hour, which is 5 beats a second.

	PENDULUM	BALANCE WHEEL AND SPRING
Restoring force	gravity	the spring
Period set by	length, and g	wheel inertia and spring stiffness
Works when moved?	no, must stay still and upright	yes, in any position
Size	about a metre for a 2 s period	millimetres to a few centimetres
Main weakness	length changes with temperature	spring stiffness and inertia change with temperature

Temperature Compensation

A new problem showed up as clocks improved: metal expands when warm. A steel rod grows by about 12 millionths of its length for each degree Celsius, and because T depends on the square root of the length, the period changes by *half* that fraction. A 10 degree summer warm-up therefore makes the clock slow by $0.5 \times 12 \times 10^{-6} \times 10 = 0.00006$, which is $86400 \times 0.00006 = 5.2$ seconds a day.

```
alpha = 12e-6              # steel, per degree C
for warmer in (1, 10):
    fraction = 0.5 * alpha * warmer
    print(warmer, "C warmer:", round(fraction * 86400, 2), "s a day slow")
# 1 C warmer: 0.52 s a day slow
# 10 C warmer: 5.18 s a day slow
```

That is small in everyday terms but large for a precise clock. The fixes came in the 1720s. George Graham made a pendulum with a jar of mercury as the bob in 1721: when the rod grows downwards, the mercury expands upwards and raises the centre of the weight by the same amount. John Harrison made a **gridiron pendulum** around 1726, built from rods of two metals, steel and brass. Brass expands more than steel, and the rods are arranged so that the brass grows upward while the steel grows downward, and the two cancel. Springs and balance wheels have their own temperature problems, which come back in Time: Clocks, Calendars and How We Measure It 8 with the clocks made for ships.

Common mistakes

Thinking a heavier bob swings slower or faster (mass does not appear in the formula); doubling the length and expecting the period to double (it grows by 1.41, the square root of 2); assuming the pendulum is exactly independent of the swing size (it is only approximately, which is the circular error); and treating g as a constant everywhere (it varies slightly with place, so a pendulum clock moved far north or south needs re-adjusting).

EXERCISE 1

Write `period(length)` and `length_for(T)` using $T = 2\pi\sqrt{(L/g)}$. Print the period for pendulums of 0.25 m, 1 m and 4 m, and the length that gives a 2 s and a 1 s period. Then find the period of the 0.994 m pendulum on the Moon ($g = 1.62$ m/s^2), and how many seconds a day a 2 s pendulum loses if it is 1 mm too long.

View solution

EXERCISE 2

Simulate a 1 m pendulum by stepping the equation of motion (angular acceleration = -g/L × sin of the angle) in tiny time steps, and time a quarter swing to find the full period for amplitudes of 2, 5, 10 and 30 degrees. Compare each with the small-swing period and with the $\theta^2/16$ formula, and give the daily error in seconds.

View solution

EXERCISE 3

Using 12 millionths per degree for steel, find the daily error of a pendulum rod that is 1 and 10 degrees warmer or colder, and the temperature range that keeps the drift within 2 seconds a day. Then, for a balance wheel beating 18000 times an hour, find the period, choose a wheel inertia of 5×10^{-8} kg m^2, work out the spring stiffness, and show how a 1 percent change in either one changes the rate.

View solution

QUICK REFERENCE

- **Pendulum period:** $T = 2\pi\sqrt{(L/g)}$; independent of the bob's mass; 0.994 m gives 2 s, 1 m gives 2.006 s
- **Galileo:** observed pendulum isochronism, around 1583 by tradition; **Huygens:** pendulum clock 1656, patent 1657, theory in 1673
- **Accuracy gain:** from about 15 minutes a day to about a minute, later about 10 seconds
- **Circular error:** period grows by about $\theta^2/16$; 5 degrees = 0.0476 percent = 41 s a day
- **Anchor escapement:** about 1670, small arc of 4 to 6 degrees; dead-beat form by Graham
- **Balance wheel and spring:** $T = 2\pi\sqrt{(I/k)}$; no g, so it works while moving
- **Temperature:** steel rod, 10 degrees warmer = 5.2 s a day slow; fixes: mercury pendulum (1721), gridiron pendulum (about 1726)

COMING NEXT

Time: Clocks, Calendars and How We Measure It 8 takes these clocks to sea: why finding longitude needed an accurate timekeeper, the rival lunar-distance method, John Harrison's H1 to H4, and worked longitude calculations.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 8 · Longitude at sea

On the night of 22 October 1707, a British fleet returning from the Mediterranean ran onto the rocks off the Isles of Scilly. Several ships were lost, including the flagship of Admiral Sir Cloudesley Shovell, and well over a thousand sailors drowned (estimates run from about 1,400 to 2,000). The fleet's navigators believed they were well out in the English Channel. They knew how far north they were, but not how far east or west. The disaster is often named as one reason Parliament took the problem seriously, and the problem was this: at sea, how do you know your longitude? The pendulums and springs of Time: Clocks, Calendars and How We Measure It 7 turned out to be part of the answer, but only after a long fight to make them work on a moving ship.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The arithmetic is exact for a spherical Earth. The history uses the commonly given dates and figures, but trial results for Harrison's clocks are quoted differently in different books, so the text says "about". The temperature figures for a steel spring are typical assumed values, not measurements of any particular watch.

Latitude Is Easy, Longitude Is Not

Latitude, your distance north or south of the equator, can be read from the sky. At noon the Sun is at its highest, and its height above the horizon, together with a table of the Sun's position for the date, gives the latitude directly. Sailors could do this with a quadrant or astrolabe for centuries. Longitude has no such marker: the stars and the Sun swing round from east to west, and nothing in the sky tells you where along that swing you are standing.

The key is the link between longitude and time, which follows from the day in Time: Clocks, Calendars and How We Measure It 1. The Earth turns 360 degrees in 24 hours, which is 360 / 24 = 15 degrees an hour. That is also 1 degree every 4 minutes. So if you know the time at a fixed reference place (the Royal Observatory at Greenwich, in the English case, founded in 1675) and the time where you are, the difference is your longitude:

TIME DIFFERENCE	LONGITUDE DIFFERENCE
1 hour	15 degrees
4 minutes	1 degree
1 minute	15 arcminutes (a quarter of a degree)
4 seconds	1 arcminute

"Local time" is easy: it is local noon when the Sun is highest. The hard part is knowing what time it is at the reference place while you are weeks away from it. A clock carried from home would do it, if it kept good time for the whole voyage. That is the idea, and it is why a chapter on longitude belongs in a course about clocks.

```
import math

def longitude(clock_hours_at_local_noon):
    # the clock keeps Greenwich time; later than 12:00 means we are west
    return (clock_hours_at_local_noon - 12.0) * 15.0

print("clock reads 14:00 at local noon:", longitude(14.0), "degrees west")
print("clock reads 15:20 at local noon:", longitude(15 + 20 / 60), "degrees west")
print("clock reads 11:20 at local noon:", longitude(11 + 20 / 60), "(negative = east)")
# clock reads 14:00 at local noon: 30.0 degrees west
# clock reads 15:20 at local noon: 50.0 degrees west
# clock reads 11:20 at local noon: -10.0 (negative = east)
```

Worked by hand: if the Greenwich clock reads 15:20 when the Sun is highest where you are, Greenwich is already 3 hours 20 minutes past its own noon. The Sun reaches its highest point in the east first and in the west later, so a place whose noon comes *later* than Greenwich's is west of it, here by 3.333 × 15 = 50 degrees.

How Much Is a Degree Worth?

A degree of longitude is not a fixed distance. At the equator, one degree is 40075 km / 360 = 111.3 km. The lines of longitude converge towards the poles, so a degree shrinks by the cosine

of the latitude: at 51.5 degrees north (London) it is 111.3 × 0.623 = 69.3 km. A clock that is a single minute wrong shifts the position by a quarter of a degree, which is 27.8 km at the equator and about 17.3 km at London's latitude.

```
equator_km = 40075
for lat in (0, 30, 51.5, 60):
    km = equator_km / 360 * math.cos(math.radians(lat))
    print("1 degree of longitude at latitude", lat, "=", round(km, 1), "km")
# 1 degree of longitude at latitude 0 = 111.3 km
# 1 degree of longitude at latitude 30 = 96.4 km
# 1 degree of longitude at latitude 51.5 = 69.3 km
# 1 degree of longitude at latitude 60 = 55.7 km
```

The Act of 1714 and the Prize

In 1714 the British Parliament passed the Longitude Act and set up a Board of Longitude to judge claims. The rewards were large for the time: £10,000 for a method accurate to 1 degree, £15,000 for 40 arcminutes (two thirds of a degree) and £20,000 for 30 arcminutes (half a degree), tested on a voyage to the West Indies. Suppose the voyage takes six weeks, 42 days. Half a degree is 30 × 4 = 120 seconds of time, so a clock aiming for the top prize may drift by only 120 / 42 = 2.9 seconds a day, steadily, through storms and from cold seas to the tropics.

```
days = 42
for name, arcmin in (("1 degree", 60), ("40 arcminutes", 40), ("30 arcminutes", 30)):
    seconds = arcmin * 4              # 1 arcminute = 4 seconds of time
    print(name, "=", seconds, "s ->", round(seconds / days, 2), "s a day allowed")
# 1 degree = 240 s -> 5.71 s a day allowed
# 40 arcminutes = 160 s -> 3.81 s a day allowed
# 30 arcminutes = 120 s -> 2.86 s a day allowed
```

Compare that with Time: Clocks, Calendars and How We Measure It 7: the best pendulum clocks of the day drifted by seconds a day on steady ground, and their rhythm depended on gravity, so the pitch and roll of a ship upset them. A ship's clock needed to be both portable and better than a good land clock.

Rival Methods: The Sky as a Clock

Not everyone thought a clock was the answer. The sky itself is a clock for anyone who knows its motions. Galileo proposed using the eclipses of Jupiter's moons, which happen at times that can be predicted and seen from anywhere. This worked on land, and was used to map coasts and territories, but it needs a long telescope held steady, which is hopeless on a rolling deck.

The more serious rival was the **method of lunar distances**. The Moon moves across the stars about 360 / 27.32 = 13.18 degrees a day (the sidereal month is 27.32 days), which is about 0.55 degrees an hour, roughly its own width. If tables predict the angle between the Moon and a bright star or the Sun for each moment of Greenwich time, a navigator can measure the angle with a sextant, look up the Greenwich time that goes with it, and compare with local time. The Moon is a hand on a giant clock face of stars. The Astronomer Royal Nevil Maskelyne began publishing the Nautical Almanac in 1767, with lunar tables that made the method practical.

```
deg_per_day = 360 / 27.32
arcmin_per_minute = deg_per_day * 60 / (24 * 60)
print(round(deg_per_day, 2), "degrees a day,", round(arcmin_per_minute, 3), "arcmin per m
minutes_wrong = 1 / arcmin_per_minute
print("1 arcmin error =", round(minutes_wrong, 2), "min of time =", round(minutes_wrong *
# 13.18 degrees a day, 0.549 arcmin per minute
# 1 arcmin error = 1.82 min of time = 0.46 degrees
```

The numbers show why this was demanding. Because the Moon moves only 0.549 arcminutes a minute, measuring its angle to one arcminute still leaves a time error of 1.82 minutes, and 1.82 minutes × 15 = 27.3 arcminutes, which is 0.46 degrees. Even a good sextant reading only just stays inside the half-degree prize limit, and the method needs a long calculation afterwards. It worked, and was used for decades, but it was slow and demanding. A chronometer needed only a glance.

	LUNAR DISTANCES	CHRONOMETER
Needs	sextant, almanac, clear sky with the Moon visible	a clock that keeps good time
Work at sea	careful measurement plus a long calculation	read the clock, note local noon, subtract
Weakness	cloud, Moon near new, skilled observer needed	clock drift, which must be checked

	LUNAR DISTANCES	CHRONOMETER
Cost	cheap instruments, a lot of training	very expensive at first

John Harrison and the Sea Clocks

John Harrison (1693 to 1776) was a Yorkshire carpenter and self-taught clockmaker. He had already built remarkable wooden clocks with his brother when he set out to make a clock for the sea. He took his ideas to London around 1730, and the first sea clock, **H1**, was finished in 1735. It was a large brass machine of about 34 kg, with two linked bars swinging in opposition (a balance with no dependence on gravity, so a ship's motion could not easily push it), springs to control the bars, and a **grasshopper escapement** that needed no oil. He also added temperature compensation made from two metals, the same idea as the gridiron pendulum in Time: Clocks, Calendars and How We Measure It 7, now in the form of a strip. In 1736 H1 went to Lisbon and back on a Royal Navy ship, and it did well enough that the Board gave Harrison money to continue.

H2 followed in 1741 and was never tested at sea, and H3 took nearly two decades and was only finished in 1759. By then Harrison had realised that the real problem was the oscillator, and he began a very different machine: **H4**, a large pocket watch about 13 cm across and only about 1.45 kg, completed in 1759. It beat five times a second, the 18000 beats an hour of Time: Clocks, Calendars and How We Measure It 7, and used a spiral balance spring, diamond pallets to cut friction, and temperature compensation built into its regulator. In 1761 H4 sailed for Jamaica in HMS *Deptford*, and after about 81 days on the voyage and return it was only about 5 seconds out. A second trial to Barbados in 1764 found it about 39 seconds out after roughly 47 days, still comfortably inside the half-degree limit.

```
equator_km = 40075

def error_km(drift_s_per_day, days):
    seconds = drift_s_per_day * days
    degrees = seconds * 15 / 3600        # 1 s of time = 15 arcseconds
    return degrees * equator_km / 360

print("1 s a day for 42 days:", round(error_km(1, 42), 1), "km at the equator")
print("10 s a day for 42 days:", round(error_km(10, 42), 1), "km")
rate = 5.1 / 81
```

```
print("H4 rate:", round(rate, 3), "s a day, 42 days:", round(error_km(rate, 42), 2), "km
# 1 s a day for 42 days: 19.5 km at the equator
# 10 s a day for 42 days: 194.8 km
# H4 rate: 0.063 s a day, 42 days: 1.23 km
```

Those numbers show the achievement. A clock drifting by even 1 second a day would put a ship about 20 km out after six weeks, and 10 seconds a day would put it out by nearly 200 km, enough to miss a harbour or hit a reef. H4's rate was 0.063 seconds a day, worth about 1.2 km. That is about 45 times better than the 2.9 seconds a day the top prize required. (As was usual with sea clocks, the figure is the rate after the clock's known steady error had been allowed for.)

Why Temperature Was the Enemy

Time: Clocks, Calendars and How We Measure It 7 promised that springs have their own temperature trouble, and here it is. A steel balance spring gets softer as it warms: its stiffness falls by roughly 0.024 percent for each degree Celsius (a typical figure, assumed here). Softer spring, longer period, slower watch. The balance wheel also expands slightly, which adds a little more. Together the loss is about 11 seconds a day for each degree, so a single degree of warming costs about four times the whole daily budget for the top prize.

```
k_change = -2.4e-4        # spring stiffness change per degree C (assumed typical)
alpha = 12e-6             # expansion of the balance, per degree C

def daily_error(delta_c):
    k_ratio = 1 + k_change * delta_c
    i_ratio = 1 + 2 * alpha * delta_c       # inertia grows as radius squared
    t_ratio = math.sqrt(i_ratio / k_ratio)
    return (t_ratio - 1) * 86400     # positive = slower

for d in (1, 10, 25):
    print(d, "C warmer: loses", round(daily_error(d), 1), "s a day")
# 1 C warmer: loses 11.4 s a day
# 10 C warmer: loses 114.2 s a day
# 25 C warmer: loses 286.4 s a day
```

This is why every successful sea timekeeper had temperature compensation. Harrison's answer used a strip of two metals bonded together, brass and steel, which bends as it warms because brass expands more. In H4 the bending strip moves a curb that changes the working length of the spring just enough to cancel the change in stiffness. Later makers such as Thomas Earnshaw and John Arnold built compensation into the balance wheel itself and used a simpler escapement, and by the early 1800s these **chronometers** were being made in numbers and carried by Navy ships, so the instrument stopped being a one-off marvel.

The Slow End of the Story

The Board of Longitude was not ready to hand over the money. It wanted the method repeated, then wanted copies of H4 built by other makers to prove the design could be reproduced, and Harrison, by then in his seventies, was also asked to explain how it worked. Larcum Kendall made a copy, known as K1, which went with Captain Cook on his second voyage (1772 to 1775) and performed well. Harrison received payments of several thousand pounds from the Board over the years, and only after the intervention of King George III in 1773 did Parliament grant him a final sum, in effect the rest of the prize. The story, often told as a lone genius against a stubborn committee, is less simple than that: the Board also paid many other people, and weighed lunar distances seriously. For some decades both methods were in use side by side.

> Common mistakes
> Reading the clock difference the wrong way (a Greenwich clock that reads *later* than 12:00 at your local noon means you are west, not east); using 111 km per degree at every latitude (it only holds at the equator, and shrinks with the cosine of the latitude); mixing up arcminutes of angle with minutes of time (1 minute of time is 15 arcminutes, 4 seconds of time is 1 arcminute); and thinking a clock must be perfectly right, when it only needs to be steady, because a known steady rate can be corrected.

EXERCISE 1

Write `longitude(hours)` that turns a Greenwich-time clock reading at local noon into degrees east or west (west if the clock reads later than 12:00). Test it with readings of 14:00, 15:20, 12:00, 11:20 and 09:40. Then work out the length of one degree of longitude at latitudes 0, 30, 51.5 and 60, and the distance that a one minute clock error would cost at the equator and at London's latitude.

View solution

EXERCISE 2

For a 42-day voyage, find the daily drift allowed by each of the three prize levels (1 degree, 40 arcminutes, 30 arcminutes). Write `error_km(drift, days)` and use it for clocks that drift by 1 and 10 seconds a day. Then take Harrison's H4 trial rate (about 5.1 seconds over 81 days), find its daily rate, its position error after 42 days and how many times better it is than the strictest prize limit.

View solution

EXERCISE 3

From the 27.32-day sidereal month, find how far the Moon moves in degrees a day and in arcminutes per minute, and how much time and longitude error a 1 arcminute mistake in a lunar distance causes. Then, with a spring stiffness falling 0.024 percent per degree Celsius and a balance expanding at 12 millionths per degree, find the daily error for 1, 10 and 25 degrees of warming, and the minutes lost over 42 days at 25 degrees.

View solution

QUICK REFERENCE

- **Longitude and time:** 15 degrees an hour, 1 degree per 4 minutes, 1 arcminute per 4 seconds of time
- **Method with a clock:** (Greenwich time at local noon − 12 h) × 15 = degrees west
- **Degree of longitude:** 111.3 km at the equator, times cos(latitude); one minute of clock error = 27.8 km at the equator
- **Longitude Act:** 1714; £10,000 (1 degree), £15,000 (40 arcminutes), £20,000 (30 arcminutes); half a degree over six weeks = 2.9 s a day
- **Lunar distances:** Moon moves 0.549 arcminute per minute; 1 arcminute error = 0.46 degrees; Nautical Almanac from 1767
- **Harrison:** H1 1735, H2 1741, H3 1759, H4 1759; H4 trial 1761-62 about 5 s in 81 days
- **Temperature:** an uncompensated steel spring loses about 11 s a day per degree; fix with a bimetallic compensator

COMING NEXT

Time: Clocks, Calendars and How We Measure It 9 follows clocks onto the land: local sun time in every town, railway timetables that forced a common time, the 1884 meridian conference, time zones and the date line, and the arguments about daylight saving.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 9 · Railways, time zones and standard time

In the 1830s, a traveller leaving London for Bristol by coach could set their watch by the church clock at either end and never think twice. The two clocks disagreed by about ten minutes, and nobody minded, because the journey took the best part of a day. Then the railway arrived and a timetable had to say when a train left one place and reached another. Which clock was it talking about? This chapter follows the answer: how a world of towns, each with its own noon, ended up with a patchwork of zones agreed by treaty, and why we still argue about moving the clocks twice a year. It builds directly on the longitude arithmetic of Time: Clocks, Calendars and How We Measure It 8, now used in the other direction.

WHAT WAS RUN, AND WHAT TO TRUST

Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. Town longitudes are rounded to two decimals, so the minutes are approximate to a few seconds. Historical dates are the commonly given ones; the number of separate railway times in North America before 1883 is quoted differently in different books, so the text says "dozens".

Every Town Its Own Noon

Before about 1840, ordinary life ran on **local mean time**: the clock in a town was set so that noon fell, on average over the year, when the Sun was highest there. (The word "mean" means that the small swings of the equation of time from Time: Clocks, Calendars and How We Measure It 2 are averaged away.) As the Earth turns 15 degrees an hour, every degree you move east or west shifts local time by 4 minutes. A town 2.59 degrees west of Greenwich, as Bristol is, has its noon 2.59 × 4 = 10.4 minutes after London's.

```
# degrees west of Greenwich (negative = east); 1 degree = 4 minutes of time
for name, west in (("Bristol", 2.59), ("Oxford", 1.26), ("Edinburgh", 3.19), ("Norwich",
    print(name, round(west * 4, 1), "min behind London")
# Bristol 10.4 min behind London
# Oxford 5.0 min behind London
# Edinburgh 12.8 min behind London
```

```
# Norwich -5.2 min behind London
```

A negative number means ahead: Norwich, to the east, saw its noon about 5 minutes before London's. For centuries this was harmless, because a journey of a day or more swallowed a gap of ten minutes without anyone noticing. Railways changed that: a train covering the same distance in a few hours made a timetable the sort of document where every minute counts.

Railway Time in Britain

The Great Western Railway, running from London to Bristol, was the first big company to settle the matter. It adopted London time along the whole line in November 1840, so a Bristol station clock and a Paddington one agreed. By the 1840s most British railway companies had followed, and in December 1847 the Railway Clearing House, the body that shared tickets and wagons between companies, recommended that all of them use Greenwich time. Town clocks were slower to give way. The Bristol Corn Exchange clock famously carried two minute hands for a while: one for local time and one for railway time.

Courts and councils were less sure, and for a time the legal answer to "what time is it?" depended on where you asked. The question was settled in Britain by the Statutes (Definition of Time) Act of 1880, which made Greenwich Mean Time the legal time for the whole of Great Britain. The telegraph helped the whole process: from 1852, a signal sent from the Royal Observatory at Greenwich could reach distant towns and railway stations at once, so a clock could be checked against it from far away.

```
gap_min = 2.59 * 4
arrive_london = 12 * 60                # train arrives at 12:00 London time
bristol_clock = arrive_london - gap_min
print("Bristol town clock reads", int(bristol_clock // 60), "h", round(bristol_clock % 6
# Bristol town clock reads 11 h 49.6 min
```

That is the puzzle a traveller faced: arrive at 12:00 by the railway timetable and the town clock outside the station said 11:49 and a bit. Travellers who trusted the wrong clock missed trains or

waited for ones that had already left.

Two Noons in North America

The trouble was far larger in North America, where distances were greater and railways were built and run by hundreds of separate companies. Each company often kept time by its own reference city, so a big station could display the times of several companies side by side. Accounts speak of dozens of different railway times in use across the continent in the early 1880s. Two engineers, Charles Dowd and the Scottish-Canadian Sandford Fleming, proposed dividing the continent into zones. In 1883 the railway companies agreed. At noon on Sunday 18 November 1883, which became known as the day of two noons, clocks across the United States and Canada were reset to one of four zones (Eastern, Central, Mountain and Pacific). Where the change put clocks back, the same few minutes were lived through twice, hence the name. The system worked well enough that it became the habit long before it became law, and the United States gave it legal force only with the Standard Time Act of 1918.

The Meridian Conference of 1884

A country could set its own zones, but what about the world? A ship's navigator and a telegraph clerk in Japan needed the same zero for longitude. In October 1884 the United States hosted the International Meridian Conference in Washington, with delegates from 25 nations. It voted to adopt a single prime meridian, the one passing through the Greenwich observatory. The vote on the key resolution was 22 in favour, 1 against (San Domingo, now the Dominican Republic), and 2 abstentions (France and Brazil). The reasons for choosing Greenwich were practical rather than scientific: by then, a large majority of the world's shipping, about two thirds of the tonnage by one count, already used charts based on Greenwich. France kept using Paris time for its railways and civil life until 1911, and even then defined its time as Paris mean time retarded by 9 minutes 21 seconds, a way of using Greenwich without saying so.

The conference also proposed a "universal day" starting at midnight at Greenwich, though it did not insist on it. That idea, a single worldwide time reference, comes back in Time: Clocks, Calendars and How We Measure It 11, where it is called UTC.

Zones: 24 Slices of the Globe

The zone idea is simple. Divide the 360 degrees of longitude by 24 hours to get 15 degrees a zone, centred on Greenwich, so Zone 0 runs from 7.5 degrees west to 7.5 degrees east. A place at longitude *L* east gets the ideal zone of the nearest multiple of 15, found by rounding *L* / 15. Inside an ideal zone the Sun clock can be at most 7.5 × 4 = 30 minutes away from the zone clock, and in practice the gap is the difference between 4 × *L* minutes and the zone's offset.

```
def ideal_zone(lon_east):
    return round(lon_east / 15)

for name, lon in (("Seattle", -122.3), ("Berlin", 13.4), ("Kolkata", 88.4)):
    z = ideal_zone(lon)
    gap = lon * 4 - z * 60        # sun time minus zone time, in minutes
    print(name, "zone", z, "sun gap", round(gap, 1), "min")
print("Kolkata on +5.5:", round(88.4 * 4 - 330, 1), "min")
# Seattle zone -8 sun gap -9.2 min
# Berlin zone 1 sun gap -6.4 min
# Kolkata zone 6 sun gap -6.4 min
# Kolkata on +5.5: 23.6 min
```

The numbers show the compromise. Here the gap is sun time minus clock time, so a negative gap means the clock runs ahead of the Sun: in Seattle, noon by the clock comes 9.2 minutes before the Sun is highest. The ideal zone for Kolkata would be 6 hours, but India chose a single national zone of 5 hours 30 minutes (written +5:30), which puts its clocks 23.6 minutes behind Kolkata's own sun time and still suits the whole country. Zones follow borders, not meridians, so a few countries use half-hour offsets (India, +5:30) or even quarter-hour ones (Nepal, +5:45), and a very wide country like China uses one zone for all of it.

	LOCAL MEAN TIME	RAILWAY TIME	STANDARD ZONES
Set by	the town's own longitude	one company or country reference	agreement between nations
Clock vs Sun	exact on average	minutes off	up to about 30 minutes, more in places
Main problem	timetables cannot work	several times in the same town	borders, offsets and the date line
Era	before about 1840	1840s to 1880s	from the 1880s onward

The Date Line

Go round the world and the zone offsets must add up to a whole day, so somewhere the date has to jump. By convention that is the **International Date Line**, which follows the 180-degree meridian in the Pacific, 12 hours from Greenwich in both directions. Cross it going west and you skip a day; cross it going east and the date goes back one day. The line is not straight: it bends around islands and countries so that neighbours stay on the same date. It has been moved more than once. Kiribati shifted it in 1994 so that all its islands share one date, and Samoa moved to the west side in 2011, skipping Friday 30 December that year so that it could trade on the same working days as Australia and New Zealand.

A flight shows the effect: leave Tokyo (9 hours ahead of Greenwich) at 17:00 on Monday and fly for 10 hours to Los Angeles (8 hours behind in winter). The clock gap is 9 + 8 = 17 hours, so you land at 17:00 + 10 h − 17 h = 10:00 the same Monday morning, seven clock hours before you left. You have not travelled in time; only the labels have moved.

Daylight Saving and the Arguments Around It

The idea of shifting the clock to make use of the morning light has an old joke at its root: in 1784 Benjamin Franklin, in a playful letter to a Paris newspaper, calculated how much candle wax Parisians could save by getting up earlier. It was a satire, not a proposal. The serious version came from a London builder, William Willett, who published *The Waste of Daylight* in 1907 and proposed moving the clocks forward by 20 minutes on each of four Sundays in April, then back in September: a total of 4 × 20 = 80 minutes. Parliament refused. It took the First World War and a drive to save coal for the idea to be adopted: Germany and Austria-Hungary started summer time on 30 April 1916, and Britain followed on 21 May 1916. Willett had died the year before.

```
sunset = 17 * 60 + 30                  # 17:30 sun time
shown = sunset + 60                      # clocks one hour forward
print("clock shows sunset at", shown // 60, ":", shown % 60)
print("daylight left after a 17:00 finish:", shown - 17 * 60, "min instead of", sunset -
# clock shows sunset at 18 : 30
# daylight left after a 17:00 finish: 90 min instead of 30
```

The daylight itself does not change; only the labels do. Moving the clock forward means the Sun rises and sets one hour later by the clock, which is why the sunset in the example moves from 17:30 to 18:30, giving an extra hour of light at the end of the working day and taking one from the morning. The arguments around it have lasted for a century. Supporters point to lighter evenings, outdoor leisure and some energy saving. Critics point to dark winter mornings for children going to school, the disruption to sleep when the clocks change, farmers whose animals ignore the clock, and studies that find the energy saving to be small or even negative. Britain tried staying on summer time all year from 1968 to 1971 and then went back to changing twice a year. In recent years many places, including parts of the European Union and several American states, have debated whether to stop changing the clocks altogether, and which of the two times to keep.

Common mistakes

Getting the direction wrong when comparing places (east of Greenwich is *ahead*, so subtract when converting back to Greenwich time); forgetting that a zone's clock and the Sun can differ by many minutes, not just a few; assuming the date line is exactly the 180-degree line when it bends; assuming a change of date is a change in time travelled; and saying that daylight saving "creates" daylight, when it only relabels the hours.

EXERCISE 1

Write a short program that turns a longitude (degrees west or east) into minutes and seconds ahead of or behind Greenwich time, and run it for Bristol (2.59 W), Oxford (1.26 W), Edinburgh (3.19 W), Exeter (3.53 W) and Norwich (1.30 E). Then work out what the Bristol town clock would show when a train arrives at 12:00 by London railway time.

View solution

EXERCISE 2

Write `ideal_zone(longitude)` and a function that gives the gap in minutes between Sun time and the zone clock. Use it for Edinburgh, New York (74.0 W), Delhi (77.2 E) and Tokyo (139.7 E). Then compare Delhi on its real +5:30 zone, and use Python's `datetime` to show a 15:00 London meeting on 14 January in New York, Delhi, Tokyo and Sydney (+11 on its summer time).

View solution

EXERCISE 3

Work out the arrival time in Los Angeles (8 hours behind Greenwich in winter) for a 10-hour flight leaving Tokyo at 17:00 on a Monday. Use `datetime` to find which weekdays Samoa went between on 30 December 2011, and how many days that year had there. Finally, work out the total of William Willett's four 20-minute steps, and the extra evening daylight when the clocks go forward an hour on a day with 17:30 sunset.

View solution

QUICK REFERENCE

- **Local mean time:** 1 degree of longitude = 4 minutes of time; Bristol is about 10.4 minutes behind London
- **Railway time:** Great Western Railway on London time from November 1840; Railway Clearing House 1847; Statutes (Definition of Time) Act 1880
- **North America:** four zones from the day of two noons, 18 November 1883; Standard Time Act 1918
- **Meridian Conference:** Washington, October 1884, 25 nations, 22 for, 1 against, 2 abstentions; Greenwich chosen as prime meridian
- **Zones:** 15 degrees each, ideal zone = round(longitude / 15); real zones follow borders and may be offset by 30 or 45 minutes
- **Date line:** near 180 degrees, bent around islands; west across it skips a day
- **Daylight saving:** Willett 1907 (4 × 20 = 80 minutes); Germany 30 April 1916, Britain 21 May 1916; it relabels hours and creates no daylight

COMING NEXT

Time: Clocks, Calendars and How We Measure It 10 goes inside the clock: the quartz crystal that runs a watch, the caesium atom that now defines the second, optical clocks, and GPS as a time service for the whole planet.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 10 · Quartz, atoms and the second

A cheap digital watch costs less than a good lunch and keeps better time than the finest brass pendulum clock of the 1800s. How did that happen? The pendulum and the balance spring of Time: Clocks, Calendars and How We Measure It 7 are both swinging objects chosen because they swing at a steady rate. This chapter follows the same idea down to its smallest and steadiest versions: a sliver of crystal that vibrates tens of thousands of times a second, then the atom itself, which vibrates billions of times a second and, unlike any man-made object, is identical wherever you find it. By the end, the second is no longer a fraction of a day but a number of atomic vibrations, and the same clocks that define it quietly run the satellite navigation in your phone.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The temperature curve for a watch crystal (about 0.034 parts per million per square degree) is a typical textbook value; real crystals vary a little from one batch to the next. Clock accuracy figures for atomic and optical clocks are the commonly quoted ones and keep improving, so treat them as "about".

A Crystal That Squeezes Out Electricity

In 1880 the French brothers Pierre and Jacques Curie found that certain crystals produce a tiny electric voltage when they are squeezed, and that the effect also works in reverse: apply a voltage and the crystal changes shape very slightly. This is called **piezoelectricity** (from the Greek for "press"). Quartz, the same mineral as ordinary sand grains but grown as a clean, pure crystal, shows it strongly.

That reversibility is the trick. Cut a quartz crystal into a small tuning-fork or bar of the right size and it rings at one very exact frequency when disturbed, like a struck bell. Its electrical signal makes it flex, its flexing makes electricity, and a simple circuit can feed that electricity back so that the crystal keeps vibrating without ever stopping. In 1927 Warren Marrison at Bell Labs in the United States used this to build the first quartz clock. Early ones filled a cabinet and needed careful temperature control, but after a few decades the electronics shrank and a quartz clock

became affordable. The first quartz wristwatch to go on sale was the Seiko Quartz Astron, in December 1969.

Inside a Quartz Watch

A modern watch has four working parts: a battery, a crystal, a small chip, and a display (or a tiny motor that turns the hands). The crystal almost always vibrates at **32,768 hertz**, which is 32,768 vibrations each second. That odd-looking number is chosen because it equals 2^{15}, and a chip can halve a frequency very cheaply with a circuit called a *divider* (a flip-flop): feed in 32,768 pulses a second and one stage gives out 16,384. Fifteen halving stages in a row turn 32,768 into exactly 1 pulse a second, and that pulse advances the seconds hand or the digits.

```
# the divider chain inside a watch
f = 32768
stages = 0
while f > 1:
    f = f // 2              # one flip-flop halves the frequency
    stages = stages + 1
print(stages, "halvings; vibrations per day:", 32768 * 86400)
# 15 halvings; vibrations per day: 2831155200
```

Nearly 2.83 billion vibrations a day, each counted by the chip without a single miss. The watch is only as good as the steadiness of that rate, which brings back the idea from Time: Clocks, Calendars and How We Measure It 1 that stability matters most.

```
watch crystal 32768 Hz
   |  divide by 2, fifteen times
   v
1 pulse a second  ----->  motor / display
   ^
 battery
```

How Good Is Quartz?

Clock errors are often quoted in **parts per million** (ppm): 1 ppm means one second gained or lost in every million seconds. A day has 86,400 seconds, so a clock that is 20 ppm fast gains 20 × 0.0864 = 1.73 seconds a day, or about 52 seconds in a 30-day month. A good watch crystal is within about 5 ppm, a little over 0.4 seconds a day, or about 13 seconds a month, which is why a typical quartz watch needs resetting once in a while but not every day.

The remaining error is mostly temperature. A tuning-fork crystal runs fastest at about 25 degrees Celsius and slows as it gets either colder or hotter, by roughly 0.034 ppm times the square of the distance in degrees from 25. At 10 degrees that is 0.034 × 15^2 = 7.65 ppm, or 0.66 seconds a day slow. Wearing a watch on the wrist keeps it near body temperature, which is why a watch on your arm often keeps better time than one on a cold shelf. Better quartz clocks put the crystal in a small heated oven, called an OCXO, so it stays at its best temperature all year, and with that they reach about a few parts in a billion. The principle is the same as the temperature compensation of Time: Clocks, Calendars and How We Measure It 7, now done by electronics instead of metal strips.

```
for temp in (25, 10, 0, 40):
    ppm = 0.034 * (temp - 25) ** 2
    print(temp, "C: slow by", round(ppm, 2), "ppm =", round(ppm * 0.0864, 2), "s a day")
# 25 C: slow by 0.0 ppm = 0.0 s a day
# 10 C: slow by 7.65 ppm = 0.66 s a day
# 0 C: slow by 21.25 ppm = 1.84 s a day
# 40 C: slow by 7.65 ppm = 0.66 s a day
```

From Crystal to Atom

Quartz crystals are all slightly different: no two are cut exactly alike, they age, and each needs its own calibration. Scientists wanted a reference that is the same everywhere in the universe. Atoms fit exactly. Every caesium-133 atom is identical to every other, so the frequency of radiation it absorbs depends only on the atom and not on how it was made.

Inside an atom the energy of an electron can only take certain fixed values. Caesium-133 has two very close energy levels in its lowest state, and it absorbs microwave radiation of one exact frequency in jumping between them. To use that as a clock, a beam of caesium atoms passes through a region filled with microwaves from a quartz-controlled source. If the microwave frequency is exactly right, the most atoms switch levels, and a detector counts them. If the frequency drifts a little, fewer atoms flip, and a feedback circuit nudges the microwave source back. The quartz oscillator provides the smooth ticking; the atoms act as the referee that keeps it honest.

```
quartz oscillator --> microwave source --> caesium atoms --> detector
       ^                                                      |
       +------------ feedback: keep the number flipping ------+
output: count the microwave cycles, and every 9,192,631,770 is one second
```

The first working caesium clock was built by Louis Essen and Jack Parry at the National Physical Laboratory in England in 1955. Its steadiness was around one part in a billion, already far better than quartz and far better than the spinning Earth itself, which is not perfectly steady. Time-keeping scientists soon decided that the second should no longer be defined from the day or the year at all.

The Second, Redefined in 1967

In 1967 the General Conference on Weights and Measures defined the SI second as **9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium-133 atom**. The number was chosen to match the length of the old second, as measured against the astronomical year, as closely as the best measurements of the day allowed, so that clocks would not suddenly disagree with the older timescale. Since then the second has been an atomic count rather than a fraction of the day.

```
caesium = 9192631770
print("one caesium period:", 1 / caesium, "s")
print("periods per watch-crystal vibration:", round(caesium / 32768, 1))
# one caesium period: 1.0878277570776665e-10 s
```

```
# periods per watch-crystal vibration: 280536.9
```

One period lasts about 0.109 nanoseconds, so 9.19 of them fit in a nanosecond. And in the time the watch crystal vibrates once, the caesium atom has gone through about 280,537 periods. A modern *caesium fountain* clock tosses cooled atoms upward, letting them rise and fall through the microwaves so that they spend longer being measured. Good fountains are accurate to roughly one part in 10^{15}, which means about one second in 30 million years.

	WATCH QUARTZ	OVEN QUARTZ	CAESIUM FOUNTAIN	OPTICAL CLOCK
Ticks at	32,768 Hz	millions of Hz	9.19 billion Hz	hundreds of trillions Hz
Typical error	about 5 ppm	few parts in 10^{9}	about 1 in 10^{15}	about 1 in 10^{18}
One second out after	about 2 days	about 10 years	about 30 million years	about 30 billion years
Found in	watches, microwaves	radio and network gear	national laboratories	research laboratories

Optical Clocks and the Next Definition

If a clock ticks faster, it can slice time more finely, and a given fractional error makes it harder to see any drift. Atoms also have transitions at the frequency of visible light, hundreds of trillions of times a second, instead of microwaves. An **optical clock** locks a laser to one of these, in a single trapped ion (aluminium or ytterbium, for example) or in thousands of strontium atoms held in a lattice of laser light. A strontium clock ticks at about 4.3×10^{14} hertz, about 46,800 times faster than caesium (4.3×10^{14} divided by 9.19×10^{9}). Over the last two decades the best optical clocks have reached uncertainties near 1 part in 10^{18}: the same clock would be a second out only after about 30 billion years, more than twice the age of the universe.

```
year = 365.25 * 86400
for e in (1e-9, 1e-15, 1e-18):
    print(e, ": one second out after", f"{(1 / e) / year:.3g}", "years")
# 1e-09 : one second out after 31.7 years
```

```
# 1e-15 : one second out after 3.17e+07 years
# 1e-18 : one second out after 3.17e+10 years
```

That is why metrologists, the people who look after units, expect to redefine the second again, probably by choosing an optical transition, in the next decade or so. The 1967 definition has served for more than half a century, but a better clock makes a better definition. Optical clocks are so sensitive that two of them placed at slightly different heights disagree slightly, because time itself runs a little differently at different heights. The reason for that is the subject of Time: Clocks, Calendars and How We Measure It 12.

GPS: A Time Service in the Sky

The Global Positioning System is a navigation system built on time. About thirty satellites circle the Earth at about 20,200 km, each carrying several atomic clocks (rubidium and caesium), and each broadcasting a message that says, roughly, "I am satellite number N, my position is here, and this is the exact time on my clock when this signal left." Your receiver compares that time with its own clock when the signal arrives. The delay is tiny: for a satellite straight overhead, 20,200 km ÷ 299,792 km a second = 0.067 seconds.

Radio waves travel at the speed of light, 299,792,458 metres a second, so a time error is a distance error. One microsecond (a millionth of a second) is 300 metres, and one nanosecond is 30 centimetres. This is the same lesson as the longitude clock of Time: Clocks, Calendars and How We Measure It 8, scaled up a million times: a good time measurement gives a position. The receiver in a phone has only a cheap quartz clock, however, which is wrong by far too much to use directly. So GPS adds a satellite. With three satellites the receiver could in principle fix its three position numbers (north, east, height). With a fourth, it can also solve for its own clock error, in effect borrowing the satellites' atomic time. A phone that has locked on to GPS therefore knows the time to within tens of billionths of a second, and many mobile networks, power grids and banking systems use GPS as their time source for exactly that reason.

```
c = 299792458
for label, t in (("1 microsecond", 1e-6), ("1 nanosecond", 1e-9)):
    print(label, "=", round(c * t, 3), "m")
print("overhead satellite delay:", round(20200e3 / c, 4), "s")
# 1 microsecond = 299.792 m
```

```
# 1 nanosecond = 0.3 m
# overhead satellite delay: 0.0674 s
```

Common mistakes

Calling a quartz watch "atomic" because the word is on the dial (only radio-controlled watches that set themselves from a time signal are linked to atomic clocks); thinking the second is defined as a fraction of the day or year (it has been an atomic count since 1967); confusing how fast a clock ticks with how accurate it is (an optical clock ticks faster than caesium and is also steadier, but a fast tick alone proves nothing); and forgetting that a one-microsecond timing error is about 300 metres of distance.

EXERCISE 1

Write a short program that counts how many times 32,768 must be halved to reach 1, and the number of crystal vibrations in a day. Work out the daily and monthly drift of a clock that is 5 ppm and one that is 20 ppm, and the ppm that equals one second a month. Finally, use the 0.034 ppm per square degree rule to find the daily drift of a watch crystal at 25, 10, 0, 35 and 40 degrees Celsius.

View solution

EXERCISE 2

Compute the length of one caesium period and how many fit in a nanosecond, and how many fit into one watch-crystal vibration. Then, for fractional errors of 1e-5, 1e-9, 1e-15 and 1e-18, work out how many years each clock takes to drift by one second, and how many times faster an optical clock at 4.3e14 hertz ticks than caesium.

View solution

EXERCISE 3

Find how far light travels in 1 microsecond, 1 nanosecond and 1 millisecond, and the signal delay from a satellite directly overhead at 20,200 km. Then build a toy one-dimensional GPS: two satellites at 0 and 40,000 km on a line, a receiver truly 12,000 km from the first whose clock is 3 ms fast. Work out the times it measures, the naive distance, and then solve for the clock error and the true position.

View solution

QUICK REFERENCE

- **Piezoelectricity:** discovered by the Curie brothers in 1880; first quartz clock by Marrison, 1927; first quartz wristwatch on sale, Seiko Astron, December 1969
- **Watch crystal:** 32,768 Hz = 2^{15}, fifteen halvings to 1 pulse a second; about 2.83 billion vibrations a day
- **Parts per million:** 1 ppm = 0.0864 seconds a day; 20 ppm = 1.73 s a day; typical crystal slows by 0.034 ppm times (temperature minus 25)^2
- **First caesium clock:** Essen and Parry, NPL, 1955
- **The SI second (1967):** 9,192,631,770 periods of the caesium-133 hyperfine transition
- **Accuracy ladder:** watch quartz about 5 ppm; caesium fountain about 1 in 10^{15}; optical clocks about 1 in 10^{18} (a second out in about 30 billion years)
- **GPS:** about 20,200 km up, atomic clocks on every satellite; 1 microsecond = 300 m, 1 nanosecond = 30 cm; four satellites give three positions plus the receiver's clock error

COMING NEXT

Time: Clocks, Calendars and How We Measure It 11 turns to computers: how a machine counts seconds from 1970, why 2038 matters, what a leap second is and why UTC has them, and how a network keeps thousands of clocks in step.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 11 · Keeping computers on time

A computer has no sense of the present. It has a counter that someone started once, a crystal that ticks it forward (the same sort of crystal as in Time: Clocks, Calendars and How We Measure It 10), and a habit of asking other machines whether it is right. Everything it "knows" about dates, from file timestamps to calendar invitations to bank transactions, is built on top of that counter. This chapter looks at how the counter is arranged, why one famous deadline in January 2038 worries engineers, why the world's civil time sometimes gains a second, how a network keeps millions of clocks in step, and why time zones are the part of the job that programmers dread most.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The history and the figures about leap seconds, the 2038 limit and the date bugs are quoted from the standard accounts; the arithmetic is worked in the text. Time zone rules change from year to year, so the outputs for 2027 reflect the rules as of this writing.

Unix Time: Seconds Since 1970

The simplest way to store a moment is as one number: how many seconds have passed since a fixed starting point. The Unix operating system, developed in the early 1970s, chose midnight at the start of 1 January 1970, UTC, called the **epoch**. A timestamp of 0 means that instant, 86,400 means a day later, and so on. A number is easy to store, compare and subtract: the gap between two events is just one value minus the other, with no months or leap years to worry about. That one idea is now everywhere, in phones, web servers and databases.

Turning the number back into a calendar date is a division. Take the timestamp 1,700,000,000. Dividing by 86,400 gives 19,675 whole days with 80,000 seconds left over. The leftover is 22 hours (79,200 seconds), 13 minutes (780 seconds) and 20 seconds, so the time of day is 22:13:20. And 19,675 days after 1 January 1970 is 14 November 2023. Together: 14 November 2023, 22:13:20 UTC.

```
from datetime import datetime, timedelta, timezone
```

```
epoch = datetime(1970, 1, 1, tzinfo=timezone.utc)
days, secs = divmod(1_700_000_000, 86400)
print(days, secs, secs // 3600, secs % 3600 // 60, secs % 60)
print(epoch + timedelta(days=days))
# 19675 80000 22 13 20
# 2023-11-14 00:00:00+00:00
```

The Year 2038 Problem

Early Unix stored the counter in a 32-bit signed whole number: 32 binary digits, one of which says plus or minus. The largest value such a number can hold is $2^{31} - 1$ = 2,147,483,647. Count that many seconds from the epoch and you reach **03:14:07 UTC on 19 January 2038**. One second later the number overflows and wraps round to −2,147,483,648, which a program reads as 20:45:52 on 13 December 1901. Systems that still keep time in a signed 32-bit field would suddenly jump back about 137 years.

```
biggest = 2 ** 31 - 1
print(biggest)
print(epoch + timedelta(seconds=biggest))
print(biggest - 1_700_000_000, "seconds left after that 2023 timestamp =",
      round((biggest - 1_700_000_000) / 86400 / 365.2425, 1), "years")
# 2147483647
# 2038-01-19 03:14:07+00:00
# 447483647 seconds left after that 2023 timestamp = 14.2 years
```

The fix is simple in principle: store the count in 64 bits. A signed 64-bit counter lasts roughly 292 billion years, many times the age of the universe, so it will not wrap in practice. Modern operating systems already use 64 bits, and the hard part is finding the old places that do not: embedded controllers in cars and factory machines, file formats and databases with 32-bit time fields, and long-lived software nobody remembers. The situation resembles the famous Year 2000 problem, where years stored as two digits made 2000 look like 1900, except that this date has been known for decades.

COUNTER	LARGEST VALUE	RUNS OUT
Signed 32-bit	2,147,483,647	19 January 2038
Unsigned 32-bit	4,294,967,295	7 February 2106
Signed 64-bit	about 9.2×10^{18}	about 292 billion years from 1970

UTC and Leap Seconds

Time: Clocks, Calendars and How We Measure It 9 described Greenwich time as the world's standard, and Time: Clocks, Calendars and How We Measure It 10 showed that the second is now an atomic count. These two ideas pull in slightly different directions. Atomic clocks tick at a perfectly steady rate. The spinning Earth does not: tides and movements inside the planet make the day vary by a few thousandths of a second, and over centuries the day has been getting a little longer. If we kept only atomic time, noon on the clock would slowly drift away from the Sun.

Coordinated Universal Time (UTC) is the compromise. It ticks at the atomic rate, but is kept within 0.9 seconds of the Earth's rotation by adding an occasional **leap second**, announced by an international body some months ahead. On the chosen day the last minute of the day, normally 60 seconds, is given 61, shown as 23:59:60. The first leap second was added on 30 June 1972, and 27 have been added altogether, the most recent at the end of 2016. On 1 January 1972, atomic time (TAI) was 10 seconds ahead of UTC, so after 27 more it is 10 + 27 = 37 seconds ahead. GPS time started on 6 January 1980, when the gap was already 19 seconds, and does not use leap seconds, so GPS time is now 37 − 19 = 18 seconds ahead of UTC.

```
leaps = 27
print("TAI minus UTC:", 10 + leaps, "s")
print("GPS minus UTC:", 10 + leaps - 19, "s")
print("seconds in a day with a leap second:", 86400 + 1)
# TAI minus UTC: 37 s
# GPS minus UTC: 18 s
# seconds in a day with a leap second: 86401
```

Leap seconds are a headache for software. Unix time pretends every day has exactly 86,400 seconds, so it has no way to write 23:59:60: the counter either repeats a second or stalls briefly, and some systems have misbehaved at a leap second. Some large operators therefore "smear" the extra second, stretching it across many hours so no clock ever sees a 61-second minute. Because of these troubles, the international measures body decided in 2022 that leap seconds should be dropped by 2035, letting UTC drift a little from the Sun instead; exactly how is still being worked out.

Drift and the Network Time Protocol

A computer's quartz clock is not very good. The crystal ranks with the cheap quartz of Time: Clocks, Calendars and How We Measure It 10, and in a warm server room it might be 10 to 50 ppm off. A 50 ppm error is 50 × 0.0864 = 4.32 seconds a day, and it also changes as the machine heats up and cools down. Left alone, a computer's clock would wander by seconds a day.

The cure is the **Network Time Protocol** (NTP), designed by David Mills in the 1980s and still in use. Machines are arranged in layers called *strata*: stratum 0 is the reference clock itself (an atomic clock or a GPS receiver), stratum 1 servers are wired directly to one, stratum 2 machines ask stratum 1, and so on. A client does not just ask "what time is it?", because the answer would be out of date by the time it arrives. Instead it records four timestamps:

```
client                        server
 t0 ---- request ---------->  t1   (server clock when it arrives)
 t3 <--- reply ------------   t2   (server clock when it leaves)

round trip   = (t3 - t0) - (t2 - t1)
clock offset = ((t1 - t0) + (t2 - t3)) / 2
```

The round trip is the time spent travelling, with the server's thinking time removed. Averaging the outward and return differences cancels the journey time, provided the trip took about the same each way, and leaves how far the client clock is from the server's. Here is an example in which the client really is half a second behind, and each leg takes 0.05 s:

```
t0, t1, t2, t3 = 100.000, 100.550, 100.551, 100.101
print(round((t3 - t0) - (t2 - t1), 3))
print(round(((t1 - t0) + (t2 - t3)) / 2, 3))
# 0.1
# 0.5
```

The sums check out: (0.550 + 0.450) / 2 = 0.5, a client half a second behind, and a round trip of 0.101 − 0.001 = 0.1 s. A client asks several servers, throws away the odd ones, and rather than jumping the clock it usually nudges it gently faster or slower, so that time never appears to run backwards. Over the internet NTP typically holds clocks to within a few milliseconds; on a local network, a fraction of a millisecond. GPS receivers do far better, to tens of nanoseconds, as Time: Clocks, Calendars and How We Measure It 10 explained, which is why phone masts and power grids use them.

TIME SOURCE	TYPICAL ACCURACY	USED BY
Uncorrected computer clock	seconds a day	nothing sensible
NTP over the internet	a few milliseconds	laptops, phones, ordinary servers
NTP on a local network	under a millisecond	offices, data centres
GPS receiver	tens of nanoseconds	phone masts, power grids, exchanges

Time Zones: The Database Behind the Names

UTC and Unix time are tidy. People are not: they want to know the local time, and Time: Clocks, Calendars and How We Measure It 9 showed how the rules came about. Every country decides its own offset from UTC and whether to use daylight saving, and they change the rules often, sometimes with only a few weeks' notice. Programmers cannot simply add a fixed number of hours, because the number depends on the place and the date.

The answer is a shared catalogue, the **time zone database** (often called tz or zoneinfo, begun in the 1980s by Arthur David Olson and now maintained as a public service by the Internet Assigned Numbers Authority). It lists every region's history of offsets and daylight saving rules, named by continent and city: `Europe/London`, `Asia/Kolkata`, `America/New_York`. It uses a city

rather than a "GMT+1" label because one offset can belong to several rule sets, and the same city may change its offset over time. Updating a phone or server often includes a fresh copy of this one file.

```
from zoneinfo import ZoneInfo

m = datetime(2027, 1, 15, 12, 0, tzinfo=timezone.utc)
for n in ("Europe/London", "Asia/Kolkata", "America/New_York"):
    print(n, m.astimezone(ZoneInfo(n)).strftime("%H:%M %Z %z"))
# Europe/London 12:00 GMT +0000
# Asia/Kolkata 17:30 IST +0530
# America/New_York 07:00 EST -0500
```

The golden rule is: **store UTC, convert only for display.** If you store a local time, you cannot always tell which moment was meant. On the night clocks go back, the hour from 01:00 to 02:00 happens twice; on the night they go forward, an hour does not exist at all. A stored "01:30" is then ambiguous or impossible. A UTC value (or a Unix timestamp) names exactly one instant and can be shown correctly anywhere. India's half-hour offset above is a reminder, too, that zones are not always whole hours (Nepal uses 45 minutes past the hour).

Classic Date Bugs

Because time is so easy to get subtly wrong, programmers keep a collection of famous failures. A few that you can work out for yourself:

BUG	WHAT WENT WRONG	THE SUM
Year 2000	years stored as two digits, so 00 looked like 1900	1900 + 0 = 1900, not 2000
Leap-year rule	"every four years" with no century exception calls 1900 a leap year (Time: Clocks, Calendars and How We Measure It 4 gave the full rule)	1900 ÷ 4 = 475, but 1900 ÷ 100 = 19 and 1900 ÷ 400 is not whole
248-day counter	a counter of hundredths of a second in a signed 32-bit number; a passenger aircraft's control units were reported in 2015 to need a restart before this limit	2^{31} ÷ 100 ÷ 86,400 = 248.55 days

BUG	WHAT WENT WRONG	THE SUM
GPS week rollover	the satellite message counts weeks in 10 binary digits, so it wraps after 1,024 weeks; receivers that did not allow for it showed the wrong date after the rollovers of 1999 and 2019	1,024 × 7 ÷ 365.24 = 19.6 years

```
print(round(2 ** 31 / 100 / 86400, 2), "days")
print(round(1024 * 7 / 365.2425, 1), "years")
# 248.55 days
# 19.6 years
```

The pattern is the same each time: a counter too small, a rule too simple, or a local time stored where a universal one was needed. Hence the standing advice to use well-tested date libraries rather than writing your own, to keep clocks synchronised, and to ask "what happens at the edges?" (midnight, the end of February, the clock change, the end of the counter).

> Common mistakes
> Saying Unix time "counts leap seconds" (it does not: every Unix day is exactly 86,400 seconds); storing local times without the zone; adding 24 hours to mean "tomorrow" when a clock change falls in between; assuming every offset is a whole number of hours; and thinking the 2038 limit affects any system that already uses 64-bit time (it does not, but older software on it may still carry 32-bit fields).

EXERCISE 1

Use Python to turn the timestamps 0, 86400, 1,000,000,000 and 1,700,000,000 into UTC dates. Find the largest signed 32-bit value and the exact UTC second it reaches, the date a wrapped counter would show one second later, and how long signed 32-bit, unsigned 32-bit and signed 64-bit counters last from 1970.

View solution

EXERCISE 2

Work out the daily drift of a 10 ppm and a 50 ppm clock and how soon each is 10 milliseconds out. Compute TAI minus UTC and GPS minus UTC from the 27 leap seconds, and the average number of

leap seconds a year from 1972 to 2016. Then apply the NTP round-trip and offset formulas to t0 = 100.000, t1 = 100.550, t2 = 100.551 and t3 = 100.101.

View solution

EXERCISE 3

Show noon UTC on 1 July 2027 in London, New York and Tokyo using `zoneinfo`. Demonstrate with UTC arithmetic the London local time that is skipped in March 2027 and the one that occurs twice in October. Then compute the overflow time of a 32-bit hundredths-of-a-second counter and write a leap-year test that gets 1900, 2000, 2024 and 2100 right.

View solution

QUICK REFERENCE

- **Unix time:** seconds since 00:00:00 UTC, 1 January 1970; every day counted as exactly 86,400 seconds
- **2038 problem:** signed 32-bit maximum 2,147,483,647 = 03:14:07 UTC, 19 January 2038; unsigned 32-bit lasts to 7 February 2106; signed 64-bit lasts about 292 billion years
- **UTC:** atomic rate, kept within 0.9 s of Earth rotation by leap seconds; first on 30 June 1972, 27 in all, last at the end of 2016; TAI − UTC = 37 s, GPS − UTC = 18 s; leap seconds are due to be dropped by 2035
- **Drift:** 1 ppm = 0.0864 s a day; 50 ppm = 4.32 s a day
- **NTP:** round trip = (t3 − t0) − (t2 − t1); offset = ((t1 − t0) + (t2 − t3)) / 2; strata 0, 1, 2…
- **Time zones:** use named zones from the tz database, never fixed offsets; store UTC and convert for display
- **Bug checklist:** small counters (248.55 days, 19.6-year GPS weeks), two-digit years, the leap-year rule, local times at clock changes

COMING NEXT

Time: Clocks, Calendars and How We Measure It 12 is the final chapter. It asks why clocks at different heights disagree, what the satellites correct for, how scientists date rocks millions of years old, and what has been proposed to replace the calendar altogether.

Time: Clocks, Calendars and How We Measure It

Course 1 · Chapter 12 · Strange time and looking ahead

For eleven chapters time has behaved itself: a day is a day, a second is a second, and two good clocks that start together stay together. This last chapter breaks that comfortable picture. Clocks at different heights disagree, the day measured by the stars is not the day measured by the Sun, rocks carry their own slow clocks measured in billions of years, and people have more than once tried to redesign the whole system. It ends with a short recap of the whole course.

> **WHAT WAS RUN, AND WHAT TO TRUST**
>
> Every Python snippet in this chapter and in the three exercise solutions was executed, and the output in the comments is what Python printed. The physical constants, half-lives and dates are quoted from standard accounts; the arithmetic is worked in the text. Orbital and other physical values are rounded.

Clocks and Gravity: Relativity in Plain Terms

In 1905 Albert Einstein showed that time is not the same for everyone: a clock that moves relative to you runs slow compared with one that stays with you. This is **time dilation**. In 1915 he extended the idea to gravity: a clock deeper in a gravitational pull runs slower than one higher up. Neither effect is noticeable in daily life, because the differences are tiny, but modern clocks are good enough to see them. In 1971 Joseph Hafele and Richard Keating flew caesium clocks round the world on airliners and found small differences from clocks left on the ground, in line with the predictions. Since 2010, optical clocks (Time: Clocks, Calendars and How We Measure It 10) have shown a change in rate when raised by only a few tens of centimetres.

Plain-language rules of thumb:

- **Speed:** a moving clock ticks slower. The slowing is about $v^2 / (2c^2)$, where v is the speed and c is the speed of light, 299,792,458 m/s.
- **Height:** a clock higher up in gravity ticks faster. Near the ground it gains about g / c^2 of a second per second for each metre of height, about 1.1×10^{-16}, or roughly 3.4 nanoseconds a year per metre.

The GPS Correction

The best everyday proof is the satellite navigation system of Time: Clocks, Calendars and How We Measure It 10. Each satellite circles about 20,200 km above the ground at roughly 3.87 km/s, carrying an atomic clock, and a position is worked out from the signal travel times. Two effects act on that clock:

- **Speed:** $(3{,}874)^2 / (2 \times (299{,}792{,}458)^2)$ = about 8.3×10^{-11}. Over a day of 86,400 s that is 7.2 microseconds *lost*.
- **Height:** the weaker gravity at orbit makes the clock run fast by about 5.3×10^{-10}, or 45.7 microseconds *gained* a day.

The net is 45.7 − 7.2 = 38.5 microseconds a day fast. That sounds like nothing, but a radio signal travels 300 metres in a microsecond, so an uncorrected satellite clock would put positions out by about 38.5 × 0.3 km = 11.5 km after a single day. The designers therefore tune each satellite's clock slightly slow before launch, so that it ticks at the right rate as seen from the ground. Satellite navigation works only because relativity is real.

```
import math

c = 299_792_458
G_M = 3.986004418e14
v = math.sqrt(G_M / 2.6560e7)
slow = v ** 2 / (2 * c ** 2)
fast = G_M / c ** 2 * (1 / 6.371e6 - 1 / 2.6560e7)
print(round(v), "m/s")
print(round((fast - slow) * 86400 * 1e6, 1), "microseconds a day")
# 3874 m/s
# 38.5 microseconds a day
```

Sidereal Time: Following the Stars

Time: Clocks, Calendars and How We Measure It 1 pointed out that a day is not the time the Earth takes to turn once. The **solar day** (noon to noon) is 24 hours; the **sidereal day** is the time for the Earth to turn once relative to the distant stars. Because the Earth also moves along its orbit, it must turn a little further each day to bring the Sun back to the same place. Over one

year of 365.2422 solar days the Earth makes one extra turn relative to the stars, so there are 366.2422 sidereal days. Each is therefore 86,400 × 365.2422 / 366.2422 = 86,164.1 s, or 23 hours 56 minutes 4.1 seconds. The stars rise about 3 minutes 56 seconds earlier each night, which is why constellations change through the year. Astronomers use sidereal time to point telescopes, because a star returns to the same spot in the sky after one sidereal day, not one solar day.

DAY	MEASURED AGAINST	LENGTH
Solar day	the Sun (noon to noon, on average)	24 h = 86,400 s
Sidereal day	the distant stars	23 h 56 min 4.1 s = 86,164.1 s
Atomic (SI) day	caesium ticks	exactly 86,400 SI seconds

Dynamical and Terrestrial Time

Astronomers who predict where planets and the Moon will be need a time that does not wobble with the Earth's spin. The old answer, in the mid-twentieth century, was **ephemeris time**, based on the motion of the planets themselves. Today it is **Terrestrial Time** (TT), which runs at the atomic rate and is fixed to atomic time by TT = TAI + 32.184 s (TAI, from Time: Clocks, Calendars and How We Measure It 11, is atomic time). Because the Earth's rotation is slowly irregular, time based on its spin (UT1) falls behind TT; the difference was about 69 seconds in the 2020s. That fits the earlier figures: TAI − UTC is 37 s, and 37 + 32.184 = 69.184 s, with UTC itself kept within 0.9 s of UT1. For calculations that need a time unaffected even by the Earth's movement round the Sun and its gravity, a further scale called Barycentric Dynamical Time is used.

Rock Clocks: Radiometric Dating

Some clocks run on a far longer scale than any tower or atomic standard. Certain atoms are unstable and turn into other atoms at a steady, unchangeable rate. The time for half of any sample to change is its **half-life**. After one half-life half remains, after two a quarter, after three an eighth, and in general the fraction left after time t is $(\tfrac{1}{2})^{t \,/\, \text{half-life}}$. Measure the fraction and you can run the formula backwards to find the age: t = half-life × $\log_2(N_0 / N)$.

Carbon-14 has a half-life of 5,730 years. Living things take in carbon continually, keeping the fraction constant; after death the carbon-14 dwindles. A bone with 25% left has passed through two half-lives, so it is 2 × 5,730 = 11,460 years old. After about ten half-lives (57,300 years) less

than a thousandth remains, so the method reaches back to roughly 50,000 years. For older rocks, **uranium-238**, with a half-life of 4.468 billion years, turns into lead-206. Comparing the uranium and lead in tiny zircon crystals gives ages in the billions of years: the oldest minerals found on Earth are about 4.4 billion years old, and meteorites give about 4.54 billion years for the Earth and the Solar System, the accepted age of our planet. Check: after 4.54 billion years the uranium remaining is $(\tfrac{1}{2})^{4.54 / 4.468} = 0.494$, so about half of the original is left, just short of one half-life.

```
import math

def age(half_life, fraction):
    return half_life * math.log2(1 / fraction)

print(round(age(5730, 0.25)), "years")
print(round(0.5 ** (4.54e9 / 4.468e9), 3))
# 11460 years
# 0.494
```

Clocks for Ten Thousand Years

At the other extreme, some people have built clocks meant to outlast civilisations. The best known is the **Clock of the Long Now**, a project of the Long Now Foundation (founded in 1996), designed by the inventor Danny Hillis and being built inside a mountain in west Texas. It is intended to run for 10,000 years. It ticks very slowly, a "century hand" advances once a century, and a cuckoo is meant to appear once a millennium. It is mechanical and made of durable materials, and it is designed to stay in step with the Sun when visitors wind it. It is a reminder that the habit of this whole course, counting repeated motions, can be pointed at long spans as well as short ones.

Remaking the Calendar and the Clock

Throughout the course we have met awkward pieces of the system: uneven months, a week that does not divide the year, 24 hours of 60 minutes. Reformers have tried to tidy them up several times, sometimes with spectacular results.

In 1793, during the French Revolution, France decreed a **decimal day**: 10 hours, each of 100 minutes, each of 100 seconds. That gives 100,000 decimal seconds a day, so a decimal second is 86,400 / 100,000 = 0.864 ordinary seconds. Three o'clock in the afternoon, 15:00, is 62.5% of the way through the day, which is 6 decimal hours 25 decimal minutes. The same revolution created a 10-day week and a calendar of twelve 30-day months plus five or six extra days. Both proved unpopular: the decimal clock was dropped in 1795 and the calendar at the start of 1806. A modern echo was "Internet Time" from a watch maker in 1998, which split the day into 1,000 "beats" of 86.4 seconds with no time zones; it did not catch on either.

Other reformers aimed at the unevenness of the months. The **International Fixed Calendar**, promoted by Moses Cotsworth in the early twentieth century, has 13 months of exactly 28 days, so 13 × 28 = 364, plus one extra "Year Day" outside the week to reach 365 (with a further Leap Day in leap years). Every month starts on the same weekday and every date always falls on the same weekday. It was used for many years inside one American photographic company, and gained supporters in the 1920s and 1930s. The **World Calendar**, put forward by Elisabeth Achelis in 1930, has twelve months arranged in four identical quarters of 91 days (31 + 30 + 30), with one extra blank day. It came near to adoption but was blocked at the United Nations in the 1950s, partly because religious groups objected to a day that would not belong to the seven-day cycle. The same objection has defeated plans that break the week, which shows how deeply a human habit like the seven-day week (Time: Clocks, Calendars and How We Measure It 5) is built into life.

PROPOSAL	IDEA	RESULT
French decimal time	10 h × 100 min × 100 s	dropped in 1795
French Republican calendar	12 months of 30 days plus 5 or 6 extra days, 10-day week	used from 1793, ended at the start of 1806
International Fixed Calendar	13 months of 28 days plus a Year Day	limited private use; never adopted by a state
World Calendar	4 equal quarters plus a blank day	rejected in the 1950s
Internet Time	1,000 beats a day, one world time	a novelty

```
fraction = (15 * 3600) / 86400
dsecs = fraction * 100_000
dh, rest = divmod(dsecs, 10_000)
```

```
dm, ds = divmod(rest, 100)
print(int(dh), int(dm), round(ds))
print(13 * 28 + 1)
# 6 25 0
# 365
```

Looking Ahead

Time-keeping has not finished changing. Leap seconds are due to be dropped by 2035; optical clocks may soon redefine the second (Time: Clocks, Calendars and How We Measure It 10); networks of ever-better clocks are being used to test gravity, search for tiny changes in the constants of nature, and measure height differences from the rate of a clock. Whether the answer is a better second, a tidier calendar or a clock that ticks once a century, the underlying question is the one that began the course: how to count something that never stops.

Course Recap

1. **Why time needs measuring:** the day, month and year do not divide evenly; a good clock is accurate, precise and stable.
2. **Shadows and water:** sundials, the equation of time and water clocks gave the first hours.
3. **Calendars from the Moon and Sun:** lunar, solar and lunisolar calendars, and intercalation to keep them in step; a year is 365.2422 days.
4. **Julian, Gregorian and the lost days:** the leap-year rules and how the calendar was corrected.
5. **Weeks, months and eras:** the seven-day week, month names, year numbering and ISO 8601.
6. **The mechanical clock:** the escapement turned falling weights into a steady tick.
7. **Pendulums and springs:** Galileo and Huygens, the balance spring and temperature compensation.
8. **Longitude and the sea clock:** a timekeeper that carries home time gives position at sea.
9. **Railways and time zones:** local time gave way to standard time, Greenwich and the date line.
10. **Quartz, atoms and the second:** the second became a count of caesium vibrations.

11. **Keeping computers on time:** Unix time, leap seconds, NTP and the time zone database.
12. **Strange time:** relativity, sidereal and dynamical time, rock clocks and reform proposals.

Common mistakes

Saying the satellite clocks "break" relativity (they confirm it); thinking the sidereal day is the "real" day and the solar day is wrong (they answer different questions); expecting carbon-14 to date very old rocks (the half-life is far too short, so uranium and similar clocks are used); and assuming a sample is gone after two half-lives (a quarter remains).

EXERCISE 1

Calculate the orbital speed of a satellite at a radius of 26,560 km, the daily slowing from its speed and the daily speeding-up from its height, and the net gain in microseconds a day. Convert that to the distance error in kilometres after one day if it were not corrected, and find how many nanoseconds a year a clock gains for each metre of height.

View solution

EXERCISE 2

Use Python to find the fraction of carbon-14 left after 5,730, 11,460, 20,000 and 50,000 years, and the age of samples that have 25% and 10% left. Then find the fraction of uranium-238 left after 1 billion years and after 4.54 billion years, and the age of a rock with 70% still intact.

View solution

EXERCISE 3

Work out the length of the sidereal day in hours, minutes and seconds from 366.2422 sidereal days in a year. Write a function that converts an ordinary time of day to French decimal time and test it on 15:00, 06:00 and 18:30. Then write a function that turns a day of the year into a month and day in the 13-month fixed calendar, and test days 1, 28, 29, 200 and 365.

View solution

QUICK REFERENCE

- **Relativity:** moving clocks run slow by about $v^2 / (2c^2)$; clocks higher in gravity run fast
- **GPS clocks:** −7.2 μs a day (speed) + 45.7 μs a day (height) = +38.5 μs a day, about 11.5 km of error a day if uncorrected
- **Sidereal day:** 86,400 × 365.2422 / 366.2422 = 86,164.1 s = 23 h 56 min 4.1 s
- **Terrestrial Time:** TT = TAI + 32.184 s
- **Half-life:** fraction left = $(½)^{t \,/\, \text{half-life}}$; age = half-life × $\log_2(N_0 / N)$; carbon-14 5,730 years, uranium-238 4.468 billion years
- **Reform proposals:** decimal second = 0.864 s; fixed calendar 13 × 28 + 1 = 365; the week has resisted every change

COURSE COMPLETE

This is the final chapter of Time: Clocks, Calendars and How We Measure It. From shadows to atoms, every method in this course has been an answer to the same problem: finding something that repeats steadily and counting it. Well done for reaching the end.