Generalitat de Catalunya
Departament d'Educació
Institut Thos i Codina

El mòdul 'datetime' de Python

Domini Públic (CC0)

Índex

1. Les classes del mòdul **1**

2. Crear dates i hores **1**
`date` . . . 1
`time` . . . 2
`datetime` . . . 2

3. Naive vs. aware: el concepte clau **2**

4. Zones horàries **3**
UTC . . . 3
Desplaçament fix . . . 3
`zoneinfo` (Python 3.9+) . . . 3
Convertir entre zones: `astimezone()` . . . 4
Convertir un naive a aware . . . 4
Canvis d'hora (DST): casos límit . . . 4

5. `timedelta`: durades i aritmètica **5**
Sumar i restar . . . 5
Diferència entre dues dates . . . 5
Aritmètica amb zones: compte amb el DST . . . 6

6. Comparar dates **6**

7. Format i anàlisi de text **7**
Format ISO 8601 (recomanat per intercanvi de dades) . . . 7
`strftime()`: de datetime a text . . . 7
`strptime()`: de text a datetime . . . 8
f-strings i `format()` . . . 8

8. Marques de temps (*timestamps*) Unix **9**

9. Informació útil d'una data **9**
Últim dia del mes i altres càlculs de calendari . . . 9

10. Exemples complets **10**
10.1. Edat a partir de la data de naixement . . . 10
10.2. Dies que falten per a una data . . . 10
10.3. Cronometrar un bloc de codi . . . 11
10.4. Llista de dies laborables d'un període . . . 11
10.5. Convertir una hora de reunió entre zones . . . 11
10.6. Fitxers de log amb data al nom . . . 12

11. Errors habituals (llista de comprovació) **12**

12. Exercicis proposats **12**

13. Resum **13**

14. Referències **13**

El mòdul `datetime` de la biblioteca estàndard serveix per treballar amb **dates, hores, instants en el temps i durades**. És una de les eines que més es fan servir (logs, caducitats, informes, planificació) i també una de les que dona més errors subtils, sobretot per les **zones horàries**. Aquest document explica com funciona i, sobretot, per què.

```
import datetime as dt
```

NOTA

Convenció habitual: `import datetime as dt` evita l'ambigüitat entre el mòdul `datetime` i la classe `datetime.datetime`.

1. Les classes del mòdul

Classe	Representa	Exemple
`dt.date`	Una data (any, mes, dia)	`2026-11-07`
`dt.time`	Una hora del dia (sense data)	`11:04:30`
`dt.datetime`	Data + hora (un instant)	`2026-11-07 11:04:30`
`dt.timedelta`	Una **durada** (diferència entre dos instants)	`3 days, 2:00:00`
`dt.timezone`	Una zona horària de desplaçament fix (p. ex. UTC+01:00)	`timezone.utc`
`dt.tzinfo`	Classe base abstracta per a zones horàries	(es fa servir via `zoneinfo`)

Tots els objectes `date`, `time`, `datetime` i `timedelta` són **immutables**: les operacions retornen objectes nous.

2. Crear dates i hores

`date`

```
d = dt.date(2026, 11, 7)
print(d)                 # 2026-11-07
print(d.year, d.month, d.day)

avui = dt.date.today()  # data local actual
```

`time`

```
t = dt.time(11, 4, 30)          # hora, minut, segon
t2 = dt.time(11, 4, 30, 500000) # + microsegons
print(t.hour, t.minute, t.second, t.microsecond)
```

`datetime`

```
x = dt.datetime(2026, 11, 7, 11, 4, 30)
print(x)                         # 2026-11-07 11:04:30

ara = dt.datetime.now()          # data i hora locals actuals
```

Combinar i separar:

```
x = dt.datetime.combine(dt.date(2026, 11, 7), dt.time(9, 0))
print(x.date())   # 2026-11-07
print(x.time())   # 09:00:00
```

Modificar camps (retorna una còpia nova):

```
y = x.replace(hour=18, minute=30)
```

3. Naive vs. aware: el concepte clau

Un `datetime` pot ser de dos tipus:

- **Naive (ingenu):** no porta cap informació de zona horària. `2026-11-07 11:04` és només un rellotge de paret: no se sap si és hora de Barcelona, de Tòquio o UTC.
- **Aware (conscient):** porta una zona horària (`tzinfo`), de manera que representa un instant **exacte** i inequívoc.

```
naive = dt.datetime(2026, 11, 7, 11, 4)
print(naive.tzinfo)      # None

aware = dt.datetime(2026, 11, 7, 11, 4, tzinfo=dt.timezone.utc)
print(aware.tzinfo)      # UTC
print(aware)             # 2026-11-07 11:04:00+00:00
```

No es poden comparar ni restar un naive amb un aware:

```
naive < aware   # TypeError: can't compare offset-naive and
 ↪  offset-aware datetimes
```

Això és un error de disseny intencionat: Python et força a decidir què vols dir en lloc d'endevinar una zona.

CONSELL

Regla pràctica: dins del programa (emmagatzematge, càlculs, comparacions) treballa amb datetimes **aware en UTC**, i converteix a hora local només per **mostrar-los** a l'usuari.

4. Zones horàries

UTC

```
ara_utc = dt.datetime.now(dt.timezone.utc)
print(ara_utc)   # 2026-11-07 10:04:30.123456+00:00
```

AVÍS

`dt.datetime.utcnow()` i `dt.datetime.utcfromtimestamp()` retornen datetimes **naive** (sense zona) tot i ser UTC, i estan **obsolets des de Python 3.12**. Fes servir `dt.datetime.now(dt.timezone.utc)` i `dt.datetime.fromtimestamp(ts, dt.timezone.utc)`.

Desplaçament fix

```
cet = dt.timezone(dt.timedelta(hours=1), "CET")
x = dt.datetime(2026, 11, 7, 11, 4, tzinfo=cet)
```

Un `timezone` de desplaçament fix **no sap res de l'horari d'estiu**. Per a zones reals cal `zoneinfo`.

`zoneinfo` (Python 3.9+)

Usa la base de dades de zones horàries IANA (`Europe/Madrid`, `America/New_York`...), que incorpora els canvis d'hora estiu/hivern:

```
from zoneinfo import ZoneInfo

madrid = ZoneInfo("Europe/Madrid")

x = dt.datetime(2026, 7, 15, 12, 0, tzinfo=madrid)
print(x)              # 2026-07-15 12:00:00+02:00   (estiu, CEST)

y = dt.datetime(2026, 11, 7, 12, 0, tzinfo=madrid)
print(y)              # 2026-11-07 12:00:00+01:00   (hivern, CET)
```

El mateix `Europe/Madrid` dona desplaçaments diferents segons la data: aquesta és la raó per la qual `timezone(timedelta(hours=1))` **no** és equivalent.

A Windows no hi ha base de dades de zones al sistema; cal instal·lar el paquet `tzdata` (`pip install tzdata`).

Convertir entre zones: `astimezone()`

```
ara_utc = dt.datetime.now(dt.timezone.utc)

print(ara_utc.astimezone(ZoneInfo("Europe/Madrid")))
print(ara_utc.astimezone(ZoneInfo("America/New_York")))
print(ara_utc.astimezone(ZoneInfo("Asia/Tokyo")))
```

`astimezone()` **conserva l'instant** i canvia la representació. Això és diferent de `replace(tzinfo=...)`, que **només enganxa una etiqueta** sense convertir:

```
x = dt.datetime(2026, 11, 7, 12, 0, tzinfo=dt.timezone.utc)

x.astimezone(madrid)          # 13:00+01:00  → mateix instant, altra
 ↪  zona
x.replace(tzinfo=madrid)      # 12:00+01:00  → un instant DIFERENT
 ↪  (error habitual)
```

Convertir un naive a aware

Si saps en quina zona va ser creat el naive:

```
naive = dt.datetime(2026, 11, 7, 11, 0)
aware = naive.replace(tzinfo=ZoneInfo("Europe/Madrid"))
```

Cridar `astimezone()` sobre un naive l'interpreta com a hora **local del sistema**, cosa que depèn de la màquina on s'executi.

Canvis d'hora (DST): casos límit

- **Hora inexistent** (primavera, se salten 02:00--03:00): `2026-03-29 02:30` a Madrid no existeix.
- **Hora ambigua** (tardor, 02:00--03:00 passa dues vegades): `2026-10-25 02:30` es produeix dues vegades. L'atribut `fold` (0 o 1) distingeix la primera i la segona.

```
x = dt.datetime(2026, 10, 25, 2, 30, tzinfo=madrid, fold=0)  # primera
 ↪  (CEST, +02:00)
y = dt.datetime(2026, 10, 25, 2, 30, tzinfo=madrid, fold=1)  # segona
 ↪  (CET, +01:00)
print(x.utcoffset(), y.utcoffset())
```

Treballar en UTC evita tots aquests problemes.

5. `timedelta`: durades i aritmètica

```
d = dt.timedelta(days=2, hours=3, minutes=30)
print(d)                  # 2 days, 3:30:00
print(d.total_seconds())  # 185400.0
```

Constructor: `weeks`, `days`, `hours`, `minutes`, `seconds`, `milliseconds`, `microseconds`. Es **normalitza** internament a `days`, `seconds` i `microseconds`.

AVÍS

Error típic: `d.seconds` **no** és la durada total en segons, sinó només la part de segons dins del dia (0--86399). Per a la durada total, fes servir `d.total_seconds()`.

```
d = dt.timedelta(days=1, hours=1)
print(d.seconds)          # 3600  (només la part sobrant del dia)
print(d.total_seconds())  # 90000.0
```

Els valors negatius també es normalitzen amb l'ull posat en el dia: `dt.timedelta(seconds=-1)` és `days=-1, seconds=86399`.

Sumar i restar

```
avui = dt.date(2026, 11, 7)

print(avui + dt.timedelta(days=30))    # 2026-12-07
print(avui - dt.timedelta(weeks=2))    # 2026-10-24

inici = dt.datetime(2026, 11, 7, 9, 0)
final = inici + dt.timedelta(hours=8, minutes=15)
```

Diferència entre dues dates

```
a = dt.date(2026, 11, 7)
b = dt.date(2027, 1, 1)

diferencia = b - a            # timedelta
print(diferencia.days)        # 55
```

Operacions admeses:

Operació	Resultat
`datetime - datetime`	`timedelta`
`datetime ± timedelta`	`datetime`
`timedelta ± timedelta`	`timedelta`
`timedelta * número` / `timedelta / número`	`timedelta`
`timedelta / timedelta`	`float`

`datetime + datetime`	**Error** (no té sentit)

Aritmètica amb zones: compte amb el DST

Quan sumes un `timedelta` a un datetime amb `ZoneInfo`, l'operació es fa sobre el **rellotge de paret**, no sobre l'instant absolut:

```
x = dt.datetime(2026, 3, 28, 12, 0, tzinfo=madrid)
y = x + dt.timedelta(days=1)
print(y)   # 2026-03-29 12:00:00+02:00 (el mateix migdia, tot i que
 ↪  han passat 23 h reals)
```

Si vols saber el temps **real** transcorregut, converteix a UTC abans de restar:

```
real = y.astimezone(dt.timezone.utc) - x.astimezone(dt.timezone.utc)
print(real)   # 23:00:00
```

6. Comparar dates

Es poden comparar amb <, <=, ==, >, >= (sempre entre objectes del mateix tipus de "consciència"):

```
a = dt.date(2026, 11, 7)
b = dt.date(2026, 12, 25)
print(a < b)                # True

# Un datetime aware es compara per instant, no per l'hora de paret
x = dt.datetime(2026, 11, 7, 12, 0, tzinfo=dt.timezone.utc)
y = x.astimezone(madrid)    # 13:00+01:00, mateix instant
print(x == y)               # True
```

Comparar un `date` amb un `datetime` és un error (`datetime` és subclasse de `date`, però la comparació llança `TypeError`). Converteix amb `.date()` si cal.

7. Format i anàlisi de text

Format ISO 8601 (recomanat per intercanvi de dades)

```
x = dt.datetime(2026, 11, 7, 11, 4, 30, tzinfo=dt.timezone.utc)

s = x.isoformat()
print(s)                    # 2026-11-07T11:04:30+00:00

y = dt.datetime.fromisoformat(s)
print(y == x)               # True
```

`fromisoformat()` accepta el format que genera `isoformat()`; des de Python 3.11 accepta també més variants d'ISO 8601, com la `Z` final. Per a formats arbitraris, usa `strptime`.

`strftime()`: de datetime a text

```
x = dt.datetime(2026, 11, 7, 11, 4, 30)

print(x.strftime("%d/%m/%Y"))          # 07/11/2026
print(x.strftime("%H:%M:%S"))          # 11:04:30
print(x.strftime("%A, %d de %B de %Y"))# Saturday, 07 de November de
 ↪  2026
```

Codis més usats:

Codi	Significat	Exemple
%Y	Any amb 4 dígits	2026
%y	Any amb 2 dígits	26
%m	Mes (01--12)	11
%d	Dia del mes (01--31)	07
%H	Hora 24 h (00--23)	11
%I	Hora 12 h (01--12)	11
%M	Minuts (00--59)	04
%S	Segons (00--59)	30
%f	Microsegons	000000
%p	AM/PM	AM
%A / %a	Nom del dia (complet/abreujat)	Saturday / Sat
%B / %b	Nom del mes (complet/abreujat)	November / Nov
%j	Dia de l'any (001--366)	311
%U / %W	Número de setmana (diumenge/dilluns inicial)	44
%z	Desplaçament UTC	+0100
%Z	Nom de la zona	CET
%%	El caràcter %	%

Noms en català: %A i %B depenen del *locale* del sistema i, per defecte, surten en anglès. Per canviar-ho:

```
import locale
locale.setlocale(locale.LC_TIME, "ca_ES.UTF-8")
print(x.strftime("%A, %d %B de %Y"))
```

Cal que el locale estigui instal·lat al sistema (`locale -a` per comprovar-ho a Linux), i `setlocale` afecta tot el procés. Una alternativa és la biblioteca externa `babel`.

NOTA

Els modificadors com `%-d` (dia sense zero inicial) funcionen a Linux/macOS, però **no són portables** a Windows (allà seria `%#d`). Si busques portabilitat, formata el número a mà: `f"{x.day}"`.

`strptime()`: de text a datetime

```
text = "07/11/2026 11:04"
x = dt.datetime.strptime(text, "%d/%m/%Y %H:%M")
print(x)   # 2026-11-07 11:04:00
```

El format ha de coincidir **exactament** amb el text, o es llança `ValueError`:

```
try:
    dt.datetime.strptime("2026-11-07", "%d/%m/%Y")
except ValueError as e:
    print(e)   # time data '2026-11-07' does not match format
     ↪  '%d/%m/%Y'
```

El resultat és **naive** tret que el format inclogui `%z`.

Per obtenir només una data:

```
d = dt.datetime.strptime("07/11/2026", "%d/%m/%Y").date()
```

f-strings i `format()`

```
x = dt.datetime(2026, 11, 7, 11, 4)
print(f"Reunió el {x:%d/%m/%Y} a les {x:%H:%M}")
# Reunió el 07/11/2026 a les 11:04
```

8. Marques de temps (*timestamps*) Unix

Un *timestamp* Unix és el nombre de segons des de l'1 de gener de 1970 a les 00:00 UTC.

```
import time

print(time.time())     # timestamp actual (float)

x = dt.datetime.now(dt.timezone.utc)
ts = x.timestamp()

# De timestamp a datetime (sempre indica la zona!)
y = dt.datetime.fromtimestamp(ts, dt.timezone.utc)
print(y == x)   # True
```

AVÍS

`datetime.timestamp()` sobre un datetime **naive** l'interpreta com a hora local del sistema. I `fromtimestamp(ts)` sense zona retorna un naive en hora local. Per evitar ambigüitats, usa sempre `tz=` explícit.

9. Informació útil d'una data

```
x = dt.date(2026, 11, 7)

print(x.weekday())      # 5   (dilluns=0 ... diumenge=6)
print(x.isoweekday())   # 6   (dilluns=1 ... diumenge=7)
print(x.isocalendar())  # (2026, 45, 6) → any ISO, setmana ISO, dia ISO
print(x.toordinal())    # nombre de dia des de l'1/1/1
print(x.isoformat())    # '2026-11-07'
```

`dt.date.min` / `dt.date.max` i `dt.datetime.min` / `dt.datetime.max` donen els límits (anys 1 a 9999).

Últim dia del mes i altres càlculs de calendari

`timedelta` no té mesos ni anys perquè **no tenen durada fixa** (28--31 dies, 365 o 366). Per a això, usa el mòdul `calendar`:

```
import calendar

any_, mes = 2026, 2
ultim_dia = calendar.monthrange(any_, mes)[1]
print(ultim_dia)                        # 28
print(calendar.isleap(2028))            # True
```

Sumar mesos a mà, sense dependències:

```
def suma_mesos(d, n):
    mes_total = d.month - 1 + n
    any_ = d.year + mes_total // 12
    mes = mes_total % 12 + 1
    dia = min(d.day, calendar.monthrange(any_, mes)[1])
    return d.replace(year=any_, month=mes, day=dia)

print(suma_mesos(dt.date(2026, 1, 31), 1))   # 2026-02-28
```

Per a regles més complexes (`+1 mes`, "últim divendres del mes", recurrències), la biblioteca externa `python-dateutil` (`relativedelta`, `rrule`) ho resol.

10. Exemples complets

10.1. Edat a partir de la data de naixement

```
import datetime as dt

def edat(naixement, avui=None):
    avui = avui or dt.date.today()
    anys = avui.year - naixement.year
    if (avui.month, avui.day) < (naixement.month, naixement.day):
        anys -= 1
    return anys

print(edat(dt.date(1980, 5, 20)))
```

10.2. Dies que falten per a una data

```
import datetime as dt

nadal = dt.date(dt.date.today().year, 12, 25)
falten = (nadal - dt.date.today()).days
print(f"Falten {falten} dies per a Nadal")
```

10.3. Cronometrar un bloc de codi

```
import datetime as dt

inici = dt.datetime.now()
# ... tasca llarga ...
durada = dt.datetime.now() - inici
print(f"Ha trigat {durada.total_seconds():.3f} s")
```

Per a mesures de rendiment precises, és millor `time.perf_counter()`, que és monòtona i no es veu afectada per canvis del rellotge del sistema.

10.4. Llista de dies laborables d'un període

```
import datetime as dt

def laborables(inici, fi):
    dia = inici
    while dia <= fi:
        if dia.weekday() < 5:       # dilluns--divendres
            yield dia
        dia += dt.timedelta(days=1)

for d in laborables(dt.date(2026, 11, 2), dt.date(2026, 11, 13)):
    print(d.strftime("%a %d/%m"))
```

10.5. Convertir una hora de reunió entre zones

```
import datetime as dt
from zoneinfo import ZoneInfo

reunio = dt.datetime(2026, 11, 12, 16, 0,
 ↪  tzinfo=ZoneInfo("Europe/Madrid"))

for zona in ["Europe/Madrid", "America/New_York",
 ↪  "America/Mexico_City", "Asia/Tokyo"]:
    local = reunio.astimezone(ZoneInfo(zona))
    print(f"{zona:22} {local:%d/%m/%Y %H:%M %Z}")
```

10.6. Fitxers de log amb data al nom

```
import datetime as dt

nom = f"backup_{dt.datetime.now():%Y%m%d_%H%M%S}.tar.gz"
print(nom)   # backup_20261107_110430.tar.gz
```

El format AAAAMMDD s'ordena alfabèticament igual que cronològicament.

11. Errors habituals (llista de comprovació)

1. **Mesclar naive i aware** → `TypeError`. Decideix una convenció (idealment, aware en UTC).
2. **Fer servir `utcnow()`** → retorna un naive. Usa `now(timezone.utc)`.
3. **Confondre `replace(tzinfo=...)` amb `astimezone(...)`**: el primer canvia l'etiqueta, el segon converteix.
4. **Usar `timedelta.seconds`** en lloc de `total_seconds()`.
5. **Format de `strptime` que no coincideix** amb el text (`%d/%m/%Y` vs. `%Y-%m-%d`), o oblidar el zero inicial en formats estrictes.
6. **Tractar un desplaçament fix com una zona real**: `timezone(timedelta(hours=1))` no canvia a l'estiu.
7. **Suposar que un dia són 24 h** en hora local: el dia del canvi d'hora en dura 23 o 25.
8. **Emmagatzemar hora local** en bases de dades: guarda UTC (o ISO 8601 amb desplaçament) i converteix en mostrar.

12. Exercicis proposats

1. Escriu un programa que demani una data de naixement en format `dd/mm/aaaa` i mostri l'edat i el dia de la setmana en què va néixer la persona.
2. Calcula quants dies falten fins al divendres vinent (si avui ja és divendres, que digui 0).
3. Fes una funció `ultim_dia_mes(any, mes)` que retorni un objecte `date`.
4. Donada una llista de timestamps en text ISO 8601 amb zones diferents, ordena-la cronològicament i mostra-la en hora de `Europe/Madrid`.
5. Comprova què passa en sumar 24 hores (`timedelta(hours=24)`) a `2026-03-28 12:00` a `Europe/Madrid`, i compara-ho amb sumar `timedelta(days=1)`. Per què el resultat és diferent si es fa a través d'UTC?
6. Crea una funció que, donada una data d'inici i una durada en dies laborables, retorni la data final sense comptar caps de setmana.
7. Genera un calendari de les 4 setmanes següents amb el nom dels dies en català usant `locale`.

13. Resum

Vull…	Eina
Data d'avui	`dt.date.today()`
Instant actual en UTC	`dt.datetime.now(dt.timezone.utc)`
Instant actual en una zona	`dt.datetime.now(ZoneInfo("Europe/Madrid"))`
Crear una data	`dt.date(a, m, d)`
Convertir entre zones	`x.astimezone(ZoneInfo(...))`
Sumar/restar temps	`x + dt.timedelta(...)`
Diferència en segons	`(b - a).total_seconds()`
Text → datetime	`dt.datetime.strptime(text, format)`
datetime → text	`x.strftime(format)` / `f"{x:%d/%m/%Y}"`
Intercanvi de dades	`x.isoformat()` / `dt.datetime.fromisoformat(s)`
Dia de la setmana	`x.weekday()` (dilluns=0)
Últim dia del mes	`calendar.monthrange(any, mes)[1]`
Timestamp Unix	`x.timestamp()` / `dt.datetime.fromtimestamp(ts, tz)`

14. Referències

- Documentació oficial de `datetime`: https://docs.python.org/es/3.15/library/datetime.html
- `zoneinfo`: https://docs.python.org/es/3.15/library/zoneinfo.html
- Codis de format de `strftime`/`strptime`: https://docs.python.org/es/3.15/library/datetime.html#strftime-and-strptime-format-codes
- `calendar`: https://docs.python.org/es/3.15/library/calendar.html
- `python-dateutil`: https://dateutil.readthedocs.io/

Versions d'aquest document

- HTML
- PDF
- ODT
- MD

Domini Públic (CC0)