```python
#!/usr/bin/env python3
"""
untis2cal — Untis-Elternsprechtag-PDF (WebUntis "Termine für den Sprechtag")
in Kalenderdateien umwandeln.

Formate:  ics  (iCalendar, mehrere Termine pro Datei; Google, Apple, Outlook, Thunderbird, ...)
          csv  (Google-Kalender-Import-Format)
          csv-outlook (Outlook-CSV, deutsche/englische Spaltennamen siehe --help)

Beispiele:
  python untis2cal.py Sprechtag.pdf                      # -> Sprechtag.ics
  python untis2cal.py Kind1.pdf Kind2.pdf -f ics csv     # mehrere Kinder, mehrere Formate
  python untis2cal.py Sprechtag.pdf --alarm 10 --dry-run # nur anzeigen, nichts schreiben
  python untis2cal.py Sprechtag.pdf --alarm 0            # ohne Erinnerung (Standard: 5 min vorher)

Benötigt:  pip install pdfplumber      (Python 3.9+)
Die PDF wird ausschließlich lokal verarbeitet.
"""
from __future__ import annotations

import argparse
import csv
import hashlib
import re
import sys
from dataclasses import dataclass, field
from datetime import date, datetime, timedelta, timezone
from pathlib import Path
from zoneinfo import ZoneInfo

import pdfplumber

TZ = ZoneInfo("Europe/Vienna")
TIME_RE = re.compile(r"^\d{1,2}:\d{2}$")


@dataclass
class Appointment:
    start: datetime
    end: datetime
    teacher: str
    subjects: list[str]
    room: str
    student: str
    school: str = ""

    def title(self, max_subjects: int = 2) -> str:
        subj = ", ".join(self.subjects[:max_subjects])
        if len(self.subjects) > max_subjects:
            subj += " u.a."
        inner = ", ".join(p for p in (subj, self.student) if p)
        return f"{self.teacher} ({inner})" if inner else self.teacher

```

```
    def description(self) -> str:
        lines = [f"Elternsprechtag – {self.teacher}"]
        if self.subjects:
            lines.append("Fächer: " + ", ".join(self.subjects))
        if self.student:
            lines.append(f"Schüler/in: {self.student}")
        if self.room:
            lines.append(f"Raum: {self.room}")
        return "\n".join(lines)

    def location(self) -> str:
        parts = [f"Raum {self.room}"] if self.room else []
        if self.school:
            parts.append(self.school)
        return ", ".join(parts)

    def uid(self) -> str:
        key = f"{self.start.isoformat()}|{self.teacher}|{self.student}"
        return hashlib.sha1(key.encode()).hexdigest()[:20] + "@untis2cal"


# ---------------------------------------------------------------------- PDF


def _cell_text(words, x_from, x_to) -> str:
    """Wörter einer Spalte zeilenweise (nach y, dann x) zusammensetzen."""
    sel = [w for w in words if x_from <= w["x0"] < x_to]
    sel.sort(key=lambda w: (round(w["top"]), w["x0"]))
    return " ".join(w["text"] for w in sel).strip()


def _tidy_name(raw: str) -> str:
    """ARTNER SABINE -> Artner Sabine; Zusätze wie HAUER_L bleiben als Hauer_L erhalten."""
    return re.sub(r"[A-ZÄÖÜ]{2,}", lambda m: m.group(0).capitalize(), raw)


def _resolve_year(day: int, month: int, school_year: str | None, explicit: int | None) -> int:
    if explicit:
        return explicit
    if school_year:
        y1, y2 = (int(x) for x in school_year.split("/"))
        return y1 if month >= 9 else y2  # Sep–Dez: 1. Jahr, Jän–Aug: 2. Jahr
    return date.today().year


def parse_pdf(path: Path, year_override: int | None = None) -> list[Appointment]:
    result: list[Appointment] = []
    with pdfplumber.open(path) as pdf:
        for page in pdf.pages:
            words = page.extract_words()
            text = page.extract_text() or ""

            m = re.search(r"Sprechtag\s+am\s+(\d{1,2})\.(\d{1,2})\.(\d{4})?", text)
            if not m:
                continue  # keine Terminseite
```

```
109                 day, month = int(m.group(1)), int(m.group(2))
110                 sy = re.search(r"Schuljahr\s*:?\s*(\d{4}/\d{4})", text)
111                 year = _resolve_year(day, month, sy.group(1) if sy else None,
112                                      int(m.group(3)) if m.group(3) else year_override)
113                 the_day = date(year, month, day)
114
115                 # Schülerkürzel: Wort mit 8-stelligem Geburtsdatum am Ende, z. B.
                    lehä20140307
116                 sm = re.search(r"\b(\S*?)\d{8}\b", text)
117                 student = sm.group(1) if sm else ""
118
119                 # Schule, z. B. "BG/BRG MÖDLING" -> "BG/BRG Mödling"
120                 school = ""
121                 sc = re.search(
                    r"\b(BG/BRG|BORG|BRG|BG|HTL|HAK|HLW|NMS|MS|VS)\s+([A-ZÄÖÜ][\wÄÖÜäöüß-]*
                    )", text)
122                 if sc:
123                     school = f"{sc.group(1)} {sc.group(2).capitalize()}"
124
125                 # Spaltengrenzen aus der Kopfzeile ableiten (robust gegen
                    Layout-Änderungen)
126                 head = {w["text"]: w for w in words if w["text"] in ("Zeitbereich",
                    "Lehrkraft", "Fächer", "Raum")}
127                 if len(head) < 4:
128                     raise ValueError(f"{path.name}: Tabellenkopf nicht gefunden")
129                 x_teacher = head["Lehrkraft"]["x0"] - 5
130                 x_subj = head["Fächer"]["x0"] - 5
131                 x_room = head["Raum"]["x0"] - 5
132                 y_head = head["Zeitbereich"]["bottom"]
133
134                 # Fußzeile abschneiden
135                 foot = [w["top"] for w in words if w["text"].startswith("WebUntis")]
136                 y_foot = (min(foot) - 20) if foot else page.height - 80
137
138                 body = [w for w in words if y_head < w["top"] < y_foot]
139
140                 # Zeilenanfänge = Uhrzeit in der ersten Spalte
141                 starts = sorted(
142                     (w for w in body if w["x0"] < x_teacher and TIME_RE.match(w["text"
                        ])),
143                     key=lambda w: w["top"],
144                 )
145                 times = [w for w in starts]
146                 # Start/Ende-Paare: jede Zeile hat zwei Zeitwörter auf gleicher Höhe
147                 rows: list[tuple[float, str, str]] = []
148                 for w in times:
149                     same = [t for t in times if abs(t["top"] - w["top"]) < 3]
150                     if w is same[0] and len(same) >= 2:
151                         same.sort(key=lambda t: t["x0"])
152                         rows.append((w["top"], same[0]["text"], same[1]["text"]))
153
154                 for i, (top, t_start, t_end) in enumerate(rows):
155                     bottom = rows[i + 1][0] - 2 if i + 1 < len(rows) else y_foot
156                     block = [w for w in body if top - 2 <= w["top"] < bottom]
157                     teacher = _tidy_name(_cell_text(block, x_teacher, x_subj))
158                     subjects_raw = _cell_text(block, x_subj, x_room)
159                     subjects = [s.strip() for s in subjects_raw.split(",") if s.strip()]
```

```
                    room = _cell_text(block, x_room, 10_000)

                    sh, sm_ = map(int, t_start.split(":"))
                    eh, em = map(int, t_end.split(":"))
                    result.append(Appointment(
                        start=datetime(the_day.year, the_day.month, the_day.day, sh,
                        sm_, tzinfo=TZ),
                        end=datetime(the_day.year, the_day.month, the_day.day, eh, em,
                        tzinfo=TZ),
                        teacher=teacher, subjects=subjects, room=room, student=student,
                         school=school,
                    ))
        return sorted(result, key=lambda a: a.start)


    # ---------------------------------------------------------------------- Writers


    def _ics_escape(s: str) -> str:
        return s.replace("\\", "\\\\").replace(";", "\\;").replace(",", "\\,").replace(
        "\n", "\\n")


    def _ics_fold(line: str) -> str:
        """Zeilen auf max. 75 Oktette falten (RFC 5545)."""
        out, cur, size = [], "", 0
        for ch in line:
            n = len(ch.encode())
            if size + n > 75:
                out.append(cur)
                cur, size = " " + ch, 1 + n
            else:
                cur += ch
                size += n
        out.append(cur)
        return "\r\n".join(out)


    def write_ics(appts: list[Appointment], path: Path, alarm_min: int | None) -> None:
        utc = lambda d: d.astimezone(timezone.utc).strftime("%Y%m%dT%H%M%SZ")
        now = datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%SZ")
        lines = ["BEGIN:VCALENDAR", "VERSION:2.0",
        "PRODID:-//untis2cal//Elternsprechtag//DE",
                 "CALSCALE:GREGORIAN", "METHOD:PUBLISH", "X-WR-CALNAME:Elternsprechtag"]
        for a in appts:
            lines += ["BEGIN:VEVENT", f"UID:{a.uid()}", f"DTSTAMP:{now}",
                      f"DTSTART:{utc(a.start)}", f"DTEND:{utc(a.end)}",
                      f"SUMMARY:{_ics_escape(a.title())}",
                      f"LOCATION:{_ics_escape(a.location())}",
                      f"DESCRIPTION:{_ics_escape(a.description())}",
                      "TRANSP:OPAQUE"]
            if alarm_min:
                lines += ["BEGIN:VALARM", "ACTION:DISPLAY", f"DESCRIPTION:{_ics_escape(
                a.title())}",
                          f"TRIGGER:-PT{alarm_min}M", "END:VALARM"]
            lines.append("END:VEVENT")
        lines.append("END:VCALENDAR")
```

```
211     path.write_bytes(("\r\n".join(_ics_fold(l) for l in lines) + "\r\n").encode(
        "utf-8"))
212
213
214 def write_csv_google(appts: list[Appointment], path: Path, alarm_min: int | None)
    -> None:
215     # Google-Kalender-Import erwartet US-Datum und 12-h-Zeit.
216     with path.open("w", newline="", encoding="utf-8-sig") as f:
217         w = csv.writer(f)
218         w.writerow(["Subject", "Start Date", "Start Time", "End Date", "End Time",
219                     "All Day Event", "Description", "Location", "Private"])
220         for a in appts:
221             w.writerow([a.title(), a.start.strftime("%m/%d/%Y"), a.start.strftime(
                "%I:%M %p"),
222                         a.end.strftime("%m/%d/%Y"), a.end.strftime("%I:%M %p"),
223                         "False", a.description(), a.location(), "True"])
224
225
226 def write_csv_outlook(appts: list[Appointment], path: Path, alarm_min: int | None)
    -> None:
227     with path.open("w", newline="", encoding="utf-8-sig") as f:
228         w = csv.writer(f)
229         w.writerow(["Subject", "Start Date", "Start Time", "End Date", "End Time",
230                     "All day event", "Reminder on/off", "Reminder Date", "Reminder
                    Time",
231                     "Description", "Location"])
232         for a in appts:
233             rem = a.start - timedelta(minutes=alarm_min) if alarm_min else None
234             w.writerow([a.title(), a.start.strftime("%d.%m.%Y"), a.start.strftime(
                "%H:%M:%S"),
235                         a.end.strftime("%d.%m.%Y"), a.end.strftime("%H:%M:%S"),
236                         "False", "True" if rem else "False",
237                         rem.strftime("%d.%m.%Y") if rem else "", rem.strftime(
                        "%H:%M:%S") if rem else "",
238                         a.description(), a.location()])
239
240
241 WRITERS = {
242     "ics": ("ics", write_ics),
243     "csv": ("google.csv", write_csv_google),
244     "csv-outlook": ("outlook.csv", write_csv_outlook),
245 }
246
247
248 # ---------------------------------------------------------------------- CLI
249
250
251 def main() -> int:
252     ap = argparse.ArgumentParser(description="Untis-Elternsprechtag-PDF →
        Kalenderdatei(en)")
253     ap.add_argument("pdf", nargs="+", type=Path, help="eine oder mehrere PDFs (z.
        B. ein PDF pro Kind)")
254     ap.add_argument("-f", "--format", nargs="+", choices=[*WRITERS, "all"], default
        =["ics"],
255                     help="Ausgabeformat(e); Standard: ics")
256     ap.add_argument("-o", "--out", type=Path, help="Ausgabe-Basisname (ohne
        Endung); Standard: erste PDF")
```

```
257         ap.add_argument("--alarm", type=int, default=5, metavar="MIN",
258                         help="Erinnerung X Minuten vorher (Standard: 5; 0 = keine Erinnerung)")
259         ap.add_argument("--year", type=int, help="Jahr erzwingen (sonst aus Schuljahr abgeleitet)")
260         ap.add_argument("--dry-run", action="store_true", help="nur anzeigen, keine Dateien schreiben")
261         args = ap.parse_args()
262 
263         appts: list[Appointment] = []
264         for p in args.pdf:
265             found = parse_pdf(p, args.year)
266             print(f"{p.name}: {len(found)} Termin(e)", file=sys.stderr)
267             appts += found
268         appts.sort(key=lambda a: a.start)
269         if not appts:
270             print("Keine Termine gefunden.", file=sys.stderr)
271             return 1
272 
273         for a in appts:
274             print(f"{a.start:%d.%m.%Y %H:%M}–{a.end:%H:%M}  {a.title():<60} {a.location()}")
275         if args.dry_run:
276             return 0
277 
278         formats = list(WRITERS) if "all" in args.format else args.format
279         base = args.out or args.pdf[0].with_suffix("")
280         for fmt in formats:
281             ext, fn = WRITERS[fmt]
282             target = Path(f"{base}.{ext}")
283             fn(appts, target, args.alarm)
284             print(f"geschrieben: {target}", file=sys.stderr)
285         return 0
286 
287 
288 if __name__ == "__main__":
289     sys.exit(main())
290 
```