```
#!/usr/bin/env python3
"""Консольный тренажёр терминов по dictionary.md рядом со скриптом."""

from __future__ import annotations

import os
import random
import re
import sys
from dataclasses import dataclass
from pathlib import Path

ROOT = Path(__file__).resolve().parent
DICTIONARY_PATH = ROOT / "dictionary.md"

HEADING_RE = re.compile(r"^(\+)?### (.+)$")
CHOICES_COUNT = 5


@dataclass
class Card:
    """Одна карточка: термин + определение + строка заголовка в файле."""

    term: str
    definition: str
    heading_line: int  # 0-based
    learned: bool


def parse_dictionary(path: Path) -> tuple[list[str], list[Card]]:
    lines = path.read_text(encoding="utf-8").splitlines()
    cards: list[Card] = []
    i = 0
    while i < len(lines):
        m = HEADING_RE.match(lines[i])
        if not m:
            i += 1
            continue
        learned = m.group(1) == "+"
        term = m.group(2).strip()
        heading_line = i
        body = []
        i += 1
        while i < len(lines) and not HEADING_RE.match(lines[i]):
            if lines[i].strip():
                body.append(lines[i].strip())
            i += 1
        definition = " ".join(body)
        if definition:
            cards.append(
                Card(
                    term=term,
                    definition=definition,
                    heading_line=heading_line,
                    learned=learned,
                )
            )
    return lines, cards


def learned_terms(cards: list[Card]) -> set[str]:
    return {c.term for c in cards if c.learned}


def active_cards(cards: list[Card], learned: set[str]) -> list[Card]:
    return [c for c in cards if c.term not in learned and not c.learned]


def mark_card_learned(lines: list[str], card: Card) -> None:
    line = lines[card.heading_line]
    if line.startswith("+###"):
        return
    if line.startswith("### "):
        lines[card.heading_line] = "+" + line


def save_dictionary(path: Path, lines: list[str]) -> None:
    path.write_text("\n".join(lines) + "\n", encoding="utf-8")
```

```
def _term_key(term: str) -> str:
    return re.sub(r"\s+", " ", term.strip()).casefold()


def unique_terms(terms: list[str]) -> list[str]:
    """Сохраняет порядок, убирает дубликаты (в т.ч. с разным регистром/пробелами)."""
    seen: set[str] = set()
    result: list[str] = []
    for term in terms:
        key = _term_key(term)
        if key not in seen:
            seen.add(key)
            result.append(term.strip())
    return result


def build_choices(correct_term: str, pool_terms: list[str], count: int = CHOICES_COUNT) -> list[str]:
    """До `count` разных терминов; если в пуле один — только правильный ответ."""
    pool = unique_terms(pool_terms)
    correct = correct_term.strip()
    correct_key = _term_key(correct)

    distractors = [t for t in pool if _term_key(t) != correct_key]
    need_wrong = min(count - 1, len(distractors))
    wrong = random.sample(distractors, need_wrong) if need_wrong else []

    options = unique_terms([correct, *wrong])
    random.shuffle(options)
    return options


def ask_int(prompt: str, lo: int, hi: int) -> int:
    while True:
        raw = input(prompt).strip()
        if not raw.isdigit():
            print(f"Введите число от {lo} до {hi}.")
            continue
        value = int(raw)
        if lo <= value <= hi:
            return value
        print(f"Введите число от {lo} до {hi}.")


def wait_to_close() -> None:
    try:
        input("\nНажмите Enter для выхода...")
    except (EOFError, KeyboardInterrupt):
        pass


def run_quiz(path: Path = DICTIONARY_PATH) -> None:
    if not path.is_file():
        print(f"Не найден словарь: {path}", file=sys.stderr)
        sys.exit(1)

    lines, cards = parse_dictionary(path)
    if not cards:
        print("Словарь пуст.")
        return

    total_terms = len({c.term for c in cards})
    mastered = learned_terms(cards)

    completed = False
    while True:
        pool = active_cards(cards, mastered)
        if not pool:
            completed = True
            break

        card = random.choice(pool)
        all_terms = unique_terms(
            [c.term for c in cards if c.term not in mastered]
        )
        options = build_choices(card.term, all_terms)
```

```
        correct_index = next(
            i for i, t in enumerate(options) if _term_key(t) == _term_key(card.term)
        )

        print("\n" + "=" * 60)
        print("Определение:")
        print(card.definition)
        print("-" * 60)
        if len(options) == 1:
            print("Остался один термин — один вариант ответа.\n")
        else:
            print("Выберите термин:\n")
        for n, term in enumerate(options, start=1):
            print(f"  {n}. {term}")

        answer = ask_int("\nВаш ответ (номер): ", 1, len(options))

        if answer - 1 == correct_index:
            print("\nВерно!")
            mark_card_learned(lines, card)
            save_dictionary(path, lines)
            card.learned = True
            mastered.add(card.term)
            remaining = total_terms - len(mastered)
            print(f"Осталось изучить: {remaining} из {total_terms}")
        else:
            print("\nНеверно.")

    if completed:
        print("\nИзучение завершено. Все термины пройдены.")
        wait_to_close()


def main() -> None:
    os.environ.setdefault("PYTHONUTF8", "1")
    for stream in (sys.stdout, sys.stdin, sys.stderr):
        try:
            stream.reconfigure(encoding="utf-8")
        except (AttributeError, OSError):
            pass
    print("Тренажёр терминов. Ctrl+C — выход.")
    run_quiz()


if __name__ == "__main__":
    main()
```