

```

#!/usr/bin/env python3
"""
utility_tools.py

A collection of small, reusable Python utilities:
- safe_read: read files safely with encoding fallback
- chunked: generator to split iterable into chunks
- timer: simple context-manager timer
- simple_http_get: perform an HTTP GET using urllib
- quicksort: example implementation
- fibonacci: iterative generator
- Example usage under __main__
"""

from __future__ import annotations
import argparse
import sys
import time
from contextlib import contextmanager
from typing import Iterable, Iterator, List, TypeVar, Generator
import urllib.request
import urllib.error

T = TypeVar("T")

def safe_read(path: str, encodings: Iterable[str] = ("utf-8", "latin-1")) -> str:
    """Read a text file trying multiple encodings and return its contents."""
    last_exc = None
    for enc in encodings:
        try:
            with open(path, "r", encoding=enc) as f:
                return f.read()
        except Exception as e:
            last_exc = e
    raise last_exc

def chunked(iterable: Iterable[T], n: int) -> Iterator[List[T]]:
    """Yield successive n-sized chunks from iterable."""
    if n <= 0:
        raise ValueError("n must be > 0")
    buf: List[T] = []
    for item in iterable:
        buf.append(item)
        if len(buf) >= n:
            yield buf
            buf = []
    if buf:
        yield buf

@contextmanager
def timer(name: str = "block"):
    """Context manager that prints elapsed time to stderr."""
    start = time.perf_counter()
    try:
        yield
    finally:
        end = time.perf_counter()
        print(f"[TIMER] {name}: {end - start:.6f}s", file=sys.stderr)

def simple_http_get(url: str, timeout: float = 10.0) -> str:
    """Perform a simple HTTP GET and return response text (utf-8)."""
    req = urllib.request.Request(url, headers={"User-Agent": "utility-tools/1.0"})
    with urllib.request.urlopen(req, timeout=timeout) as resp:
        raw = resp.read()
        try:
            return raw.decode("utf-8")
        except Exception:
            return raw.decode("latin-1", errors="replace")

def quicksort(a: List[T]) -> List[T]:

```

```

"""Simple quicksort (not in-place) for demonstration."""
if len(a) <= 1:
    return a[:]
pivot = a[len(a) // 2]
left = [x for x in a if x < pivot]
middle = [x for x in a if x == pivot]
right = [x for x in a if x > pivot]
return quicksort(left) + middle + quicksort(right)

def fibonacci(n: int) -> Generator[int, None, None]:
    """Yield first n Fibonacci numbers (0-based)."""
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

def parse_args(argv=None):
    p = argparse.ArgumentParser(description="Utility tools demo")
    p.add_argument("--fib", type=int, help="print first N Fibonacci numbers")
    p.add_argument("--fetch", type=str, help="perform a simple HTTP GET and print length")
    p.add_argument("--file", type=str, help="print the first 400 characters of a file")
    p.add_argument("--sort", nargs="+", help="sort the provided list of numbers", default=[])
    return p.parse_args(argv)

def main(argv=None):
    args = parse_args(argv)
    if args.fib is not None:
        print(*".join(str(x) for x in fibonacci(args.fib)))")
    if args.fetch:
        with timer("fetch"):
            body = simple_http_get(args.fetch)
            print(f"Fetched {len(body)} bytes from {args.fetch}")
    if args.file:
        try:
            text = safe_read(args.file)
            print(text[:400])
        except Exception as e:
            print("Error reading file:", e, file=sys.stderr)
    if args.sort:
        try:
            nums = [float(x) for x in args.sort]
            sorted_nums = quicksort(nums)
            print(*".join(str(x) for x in sorted_nums))")
        except Exception as e:
            print("Error sorting inputs:", e, file=sys.stderr)

if __name__ == "__main__":
    main()

```