Email Sequence Averager

User Handbook • v2.0

This handbook explains how to configure, run, and maintain the **email_sequence_averager.py** script. It also includes a prompt template you can use to regenerate the script with different settings.

1. What the Script Does

The script connects to your email inbox via IMAP, scans incoming messages, and extracts **27-digit sequences** of integers (each digit is 0, 1, 2, or 3) wrapped in the markers **VfE**…**CAR**.

Example of a valid sequence in an email body:

```
VfE102301231023010231021302310CAR
```

Rules applied:

- Only emails from **pre-approved sender domains** are processed. All others are silently skipped.
- Each unique sender address gets exactly **one vote**. If a sender submits multiple emails, only their most recent sequence (by email Date header) counts.
- The script averages all current sequences **position-by-position**, rounding each result to the nearest integer (0, 1, 2, or 3).
- Both individual sequences and the averaged result are framed as **VfE…CAR** in the output file.

State is saved between runs, so the average always reflects the full history of the latest submission from each sender.

2. Requirements

- **Python 3.10** or later
- **No third-party packages** — only Python standard library modules are used (imaplib, email, re, json, sys, datetime, pathlib).
- An email account accessible via **IMAP with SSL** (port 993 is the standard).

3. Files Created by the Script

email_sequence_averager.py

The main script. Edit the configuration block at the top before the first run.

sequences_state.json

Auto-created on the first run. Stores the latest sequence and timestamp for every accepted sender address. **Delete this file to reset all history.**

sequences_output.txt

Auto-created on the first run. A timestamped report is appended every time the script runs.

4. Configuration

Open **email_sequence_averager.py** in any text editor and edit the block near the top of the file:

IMAP_SERVER

The hostname of your mail server. Common examples:

```
imap.gmail.com           # Gmail
outlook.office365.com    # Microsoft 365
imap.mail.yahoo.com      # Yahoo
imap.yourcompany.com     # Custom / corporate
```

IMAP_PORT

Almost always **993** (SSL). Do not change unless your provider explicitly uses a different port.

IMAP_USERNAME

Your full email address, e.g. *yourname@yourcompany.com*.

IMAP_PASSWORD

Your account password or, preferably, an app-specific password (see Section 6 — Security).

ALLOWED_DOMAINS

A Python list of domains from which sequences are accepted. Emails from any other domain are ignored entirely.

```
ALLOWED_DOMAINS = [
    "yourcompany.com",
    "trusted-partner.org",
]
```

IMAP_FOLDER

The mailbox folder to scan. Default: **"INBOX"**. Change if sequences arrive in a sub-folder, e.g. *"Sequences"*.

OUTPUT_FILE

Path and filename of the results log. Default: **"sequences_output.txt"**.

STATE_FILE

Path and filename of the persistent state store. Default: **"sequences_state.json"**.

5. How to Run

Step 1 — Open a terminal

Command Prompt or PowerShell on Windows; Terminal on macOS or Linux.

Step 2 — Navigate to the script folder

```
cd /path/to/your/folder
```

Step 3 — Run the script

```
python email_sequence_averager.py
# or on some systems:
python3 email_sequence_averager.py
```

Step 4 — Watch the live log

```
Loading previous state...
  3 sender(s) on record.
Connecting to IMAP server and scanning emails...
  [SKIP] Ignored domain: unknown.net (x@unknown.net)
  [NEW]  alice@example.com -> VfE102...CAR
  [UPD]  bob@example.com -> VfE231...CAR
  [OLD]  Skipped older email from bob@example.com
Done. Results written to 'sequences_output.txt'.
Averaged sequence (3 sender(s)): VfE112...CAR
```

Step 5 — Read the report

Open **sequences_output.txt** to see the full timestamped report.

> *To automate, schedule the script with cron (macOS/Linux) or Task Scheduler (Windows).*

6. Security Recommendations

Storing a plain-text password in the script is convenient but not secure. Better alternatives:

Option A — Environment variable (recommended)

Replace the IMAP_PASSWORD line in the script with:

```
import os
IMAP_PASSWORD = os.environ.get("IMAP_PASSWORD", "")
```

Then set the variable before running:

```
export IMAP_PASSWORD="your_password"   # macOS / Linux
set IMAP_PASSWORD=your_password        # Windows CMD
```

Option B — App-specific password

Gmail, Outlook, and Yahoo support app-specific passwords that can be revoked independently of your main password. Enable 2-factor authentication, then generate an app password from your account security settings.

Option C — Isolate with a sub-folder

Create a dedicated mailbox folder (e.g. "Sequences"), set up an email filter to auto-move submissions there, and point IMAP_FOLDER at that folder so the script never touches the rest of your inbox.

7. Understanding the Output File

Each run appends a block similar to the following:

```
============================================================
Run timestamp : 2024-05-01T14:32:10.123456
Senders with a sequence on record : 3
Senders updated this run          : 1
------------------------------------------------------------
Updates this run (latest sequence per sender):

  Sender    : alice@example.com
  Email date: 2024-05-01T12:00:00+00:00
  Previous  : VfE1023012310230102310213023l0CAR
  New latest: VfE2103120312031203120312031203021CAR

------------------------------------------------------------
Current latest sequence per sender:
  alice@example.com       VfE2103120312031203120312031203021CAR
  bob@trusted.org         VfE0123012301230123012301230123012CAR
  carol@example.com       VfE1230123012301230123012301230123CAR
------------------------------------------------------------
Cumulative averaged sequence (one vote per sender):
  VfE1120120311230113120312020l1CAR
============================================================
```

8. Resetting the State

Reset all history (start fresh)

Delete **sequences_state.json**. The script will recreate it on the next run and treat every email as a first-time submission.

Clear the output log without losing state

Delete **sequences_output.txt**. The script will recreate it on the next run.

9. Troubleshooting

"Could not connect to IMAP server"

- Check IMAP_SERVER and IMAP_PORT.
- Ensure IMAP access is enabled in your mail account settings. Gmail: Settings → See all settings → Forwarding and POP/IMAP → Enable IMAP.
- Check that your firewall or VPN is not blocking port 993.

"Login failed"

- Double-check IMAP_USERNAME and IMAP_PASSWORD.

- Gmail/Outlook may require an app-specific password when 2-factor authentication is active.

"No sequences found" despite valid emails

- Confirm the email body contains the exact markers **VfE** (capital V, lowercase f, capital E) and **CAR** (all capitals).
- Only digits 0, 1, 2, and 3 are valid. A digit of 4 or higher breaks the match.
- Digits may be separated by spaces or commas, but the total count must be exactly 27.

"[OLD] Skipped older email"

This is expected behaviour. The script already holds a newer sequence for that sender address. Only the most recent submission per address counts toward the average.

Emails from allowed domains still skipped

- Verify the From header contains the full address. Some mailing systems use display names only.
- *mail.example.com* does NOT match *example.com*. Add sub-domains explicitly if needed.

10. Changing the Initial Settings

If you want a different version of the script — for example, using a different email protocol, output method, or averaging strategy — use the prompt template below when starting a new conversation with an AI assistant such as Claude.

Prompt Template for Regenerating the Script

```
Please create a Python script that processes email sequences
with the following specifications:

EMAIL ACCESS
  Protocol : [IMAP / Gmail API / Microsoft Graph API
              / Parse local .eml files]
  Server   : [hostname, if IMAP]
  Port     : [993 for SSL IMAP, or as required]

SEQUENCE FORMAT
  Markers         : VfE (start) and CAR (end)
  Sequence length : 27 digits
  Valid digits    : 0, 1, 2, 3
  Averaging rule  : round each position to nearest int

SENDER FILTERING
  Allowed domains : [list your domains here]
  Per-sender rule : keep only the latest submission
                    per unique sender address

OUTPUT
  Method  : [Write to file / Print to terminal /
```

```
             Send reply email / All of the above]
  Format  : append a timestamped report each run;
            show per-sender latest sequences and the
            cumulative averaged sequence as VfE...CAR

AVERAGING SCOPE
  [Average all emails ever received (cumulative) /
   Average only emails received since last run /
   Average a fixed batch of N emails]

STATE PERSISTENCE
  [Yes - save state between runs in a JSON file /
   No  - process fresh each run]

SECURITY
  [Store password in script / Use environment variable
   / Use app-specific password flow]

Additional requirements:
  [add here, e.g. scheduled automation, sub-folder
   isolation, logging level, error notifications, etc.]
```

Fill in the bracketed fields and paste the completed prompt to get a script tailored to your new settings.

IMPORTANT — SENDER APP REQUIREMENTS

Users who generate and send sequences (i.e. the senders of incoming mail processed by this script) must be running a supported version of the **VISTALIZER® for Enterprises** app.

Minimum required versions:

iOS / iPadOS Version 3.6.2 or later

Android Build 173 (version 1.7.3) or later

Sequences generated by older versions of the app may not conform to the expected VfE…CAR format and will be rejected or silently ignored by the script. Advise all participants to update their app before submitting sequences.

Appendix — Full Script Listing

email_sequence_averager.py

```
#!/usr/bin/env python3
"""
Email Sequence Averager
-----------------------
Connects to an IMAP mailbox, extracts 27-integer sequences (values 0–3)
framed by VfE...CAR from allowed sender domains.

Only the LATEST sequence submitted by each individual sender address is
kept. The cumulative average is computed across one sequence per sender,
rounded per position to the nearest integer, and written to an output file.
"""

import imaplib
import email
import email.utils
import re
import json
import sys
from datetime import datetime, timezone
from pathlib import Path

# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# CONFIGURATION — edit these before running
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

IMAP_SERVER   = "imap.example.com"   # e.g. "imap.gmail.com"
IMAP_PORT     = 993                  # 993 for SSL
IMAP_USERNAME = "your@email.com"
IMAP_PASSWORD = "your_password"      # consider using an app password or env var

# Only emails from these domains will be processed
ALLOWED_DOMAINS = [
    "example.com",
    "trusted-partner.org",
    # add more domains here
]

# Folder to check (usually "INBOX")
IMAP_FOLDER = "INBOX"

# Output file where sequences and the average are saved
OUTPUT_FILE = "sequences_output.txt"

# State file — stores the latest sequence per sender address
STATE_FILE = "sequences_state.json"

# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# SEQUENCE PATTERN
# Matches: VfE followed by exactly 27 digits (0-3), then CAR
# Digits may be separated by optional whitespace or commas
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

RAW_PATTERN = re.compile(
    r"VfE\s*((?:[0-3][\s,]*){27})\s*CAR"
)

SEQUENCE_LENGTH = 27
```

```
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# HELPERS
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def extract_sequences_from_text(text: str) -> list[list[int]]:
    """Find all VfE...CAR sequences in a block of text."""
    sequences = []
    for match in RAW_PATTERN.finditer(text):
        digits = re.findall(r"[0-3]", match.group(1))
        if len(digits) == SEQUENCE_LENGTH:
            sequences.append([int(d) for d in digits])
    return sequences


def parse_sender(from_header: str) -> tuple[str, str]:
    """
    Return (display_name, address) from a From header.
    Address is lowercased.
    """
    name, addr = email.utils.parseaddr(from_header)
    return name, addr.lower()


def get_domain(address: str) -> str:
    """Extract domain from a full email address."""
    return address.split("@")[-1] if "@" in address else ""


def parse_date(date_header: str) -> datetime:
    """
    Parse an email Date header into a timezone-aware datetime.
    Falls back to epoch on failure.
    """
    try:
        timestamp = email.utils.mktime_tz(email.utils.parsedate_tz(date_header))
        return datetime.fromtimestamp(timestamp, tz=timezone.utc)
    except Exception:
        return datetime.fromtimestamp(0, tz=timezone.utc)


# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# IMAP
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def fetch_emails(imap_server, port, username, password, folder):
    """
    Connect via IMAP and yield (sender_address, sent_datetime, body_text)
    for every email in the folder, ordered oldest-first (IMAP default).
    """
    try:
        mail = imaplib.IMAP4_SSL(imap_server, port)
    except Exception as e:
        print(f"[ERROR] Could not connect to IMAP server: {e}")
        sys.exit(1)

    try:
        mail.login(username, password)
    except imaplib.IMAP4.error as e:
        print(f"[ERROR] Login failed: {e}")
        sys.exit(1)

    mail.select(folder)
```

```
    _, message_ids = mail.search(None, "ALL")

    for msg_id in message_ids[0].split():
        _, msg_data = mail.fetch(msg_id, "(RFC822)")
        raw = msg_data[0][1]
        msg = email.message_from_bytes(raw)

        _, sender_address = parse_sender(msg.get("From", ""))
        sent_dt = parse_date(msg.get("Date", ""))

        # Gather plain-text body
        body_parts = []
        if msg.is_multipart():
            for part in msg.walk():
                if part.get_content_type() == "text/plain":
                    try:
                        body_parts.append(
                            part.get_payload(decode=True).decode(errors="replace")
                        )
                    except Exception:
                        pass
        else:
            try:
                body_parts.append(
                    msg.get_payload(decode=True).decode(errors="replace")
                )
            except Exception:
                pass

        yield sender_address, sent_dt, "\n".join(body_parts)

    mail.logout()


# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# STATE  (keyed by sender address)
#
# Structure stored in JSON:
# {
#   "alice@example.com": {
#       "sequence": [1, 0, 2, ...],   # 27 ints
#       "timestamp": "2024-05-01T12:00:00+00:00"
#   },
#   ...
# }
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def load_state(state_file: str) -> dict:
    """Load per-sender state from disk."""
    if Path(state_file).exists():
        try:
            with open(state_file, "r") as f:
                return json.load(f)
        except (json.JSONDecodeError, ValueError):
            return {}
    return {}


def save_state(state_file: str, state: dict):
    """Persist per-sender state to disk."""
    with open(state_file, "w") as f:
        json.dump(state, f, indent=2)
```

```
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# AVERAGING
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def compute_average(state: dict) -> list[int]:
    """
    Average per-position across the latest sequence of every sender,
    rounded to the nearest integer.
    """
    sequences = [entry["sequence"] for entry in state.values()]
    totals = [0] * SEQUENCE_LENGTH
    for seq in sequences:
        for i, val in enumerate(seq):
            totals[i] += val
    n = len(sequences)
    return [round(t / n) for t in totals]


def format_sequence(seq: list[int]) -> str:
    """Wrap a sequence with VfE prefix and CAR suffix."""
    return "VfE" + "".join(str(d) for d in seq) + "CAR"


# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# OUTPUT
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def write_output(
    output_file: str,
    updates: list[dict],   # list of {sender, old_seq, new_seq, timestamp}
    state: dict,
    averaged: list[int],
):
    """Append a run report to the output file."""
    with open(output_file, "a", encoding="utf-8") as f:
        f.write("=" * 60 + "\n")
        f.write(f"Run timestamp : {datetime.now().isoformat()}\n")
        f.write(f"Senders with a sequence on record : {len(state)}\n")
        f.write(f"Senders updated this run          : {len(updates)}\n")
        f.write("-" * 60 + "\n")

        if updates:
            f.write("Updates this run (latest sequence per sender):\n")
            for u in updates:
                f.write(f"\n  Sender    : {u['sender']}\n")
                f.write(f"  Email date: {u['timestamp']}\n")
                if u["old_seq"]:
                    f.write(f"  Previous  : {format_sequence(u['old_seq'])}\n")
                else:
                    f.write(f"  Previous  : (none — first submission)\n")
                f.write(f"  New latest: {format_sequence(u['new_seq'])}\n")
        else:
            f.write("No updates — no new sequences found in this run.\n")

        f.write("\n" + "-" * 60 + "\n")
        f.write("Current latest sequence per sender:\n")
        for sender, entry in sorted(state.items()):
            f.write(f"  {sender:<35} {format_sequence(entry['sequence'])}\n")

        f.write("-" * 60 + "\n")
        f.write("Cumulative averaged sequence (one vote per sender):\n")
        f.write(f"  {format_sequence(averaged)}\n")
```

```
        f.write("=" * 60 + "\n\n")


# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
# MAIN
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

def main():
    print("Loading previous state...")
    state = load_state(STATE_FILE)
    print(f"  {len(state)} sender(s) on record.")

    print("Connecting to IMAP server and scanning emails...")
    updates = []

    for sender_address, sent_dt, body in fetch_emails(
        IMAP_SERVER, IMAP_PORT, IMAP_USERNAME, IMAP_PASSWORD, IMAP_FOLDER
    ):
        domain = get_domain(sender_address)
        if domain not in ALLOWED_DOMAINS:
            print(f"  [SKIP] Ignored domain: {domain} ({sender_address})")
            continue

        sequences = extract_sequences_from_text(body)
        if not sequences:
            print(f"  [--]   No sequence found from {sender_address}")
            continue

        # Take only the last sequence found in this particular email
        new_seq = sequences[-1]
        sent_iso = sent_dt.isoformat()

        existing = state.get(sender_address)

        # Update only if this email is newer than what we already have
        if existing is None or sent_iso > existing["timestamp"]:
            old_seq = existing["sequence"] if existing else None
            state[sender_address] = {
                "sequence": new_seq,
                "timestamp": sent_iso,
            }
            updates.append({
                "sender": sender_address,
                "old_seq": old_seq,
                "new_seq": new_seq,
                "timestamp": sent_iso,
            })
            tag = "NEW" if old_seq is None else "UPD"
            print(f"  [{tag}]   {sender_address} -> {format_sequence(new_seq)}")
        else:
            print(f"  [OLD]   Skipped older email from {sender_address} (have newer)")

    if not state:
        print("No sequences on record. Exiting.")
        return

    save_state(STATE_FILE, state)

    averaged = compute_average(state)
    write_output(OUTPUT_FILE, updates, state, averaged)

    print(f"\nDone. Results written to '{OUTPUT_FILE}'.")
```

```
    print(f"Averaged sequence ({len(state)} sender(s)): {format_sequence(averaged)}")


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

End of Handbook