Tutorial

Hello, in this document, I will show you the step-by-step process for archiving the entire forum in the event of an apocalypse.
Realistically, this will take 1-3 days to download
(Made by pio)

Step 1.

Install your Python dependencies.
https://www.python.org/downloads/ + make sure to check **"Add Python to PATH"** to download **pip**.

Run in command center/terminal: "**pip install requests beautifulsoup4 cloudscraper**"

Step 2.

Save this script as "looksmax_archiver.py"
https://gofile.io/d/mX9bpJ
In case the gofile link expires ill paste the full script in another tab. (Just paste it into a .txt and rename it to looksmax_archiver.py)

Now run the script: "python looksmax_archiver.py"

Step 3.

It will run automatically and:

Discover all forums from the homepage, it will go through every page, through every thread listing. Scrape every post on every thread page. Save everything to looksmax_archive.db (SQLite)

- **Resume support** — if it crashes or you stop it, just re-run and it picks up exactly where it left off. Nothing is re-scraped twice.
- **Cloudflare bypass** — uses cloudscraper which handles the JS challenge automatically
- **Polite delays** — 2–4 second random waits between requests so you don't get IP banned
- **Retry logic** — exponential backoff on failed requests
- **Logging** — writes to both terminal and archiver.log

looksmax_archiver.py

```
"""
looksmax.org Full Archive Scraper
==================================
Scrapes all forums, threads, and replies from looksmax.org (XenForo).
Saves everything to a local SQLite database + optional HTML files.

Requirements:
    pip install requests beautifulsoup4 cloudscraper

Usage:
    python looksmax_archiver.py

Output:
    - looksmax_archive.db  (SQLite database with all content)
    - pages/               (optional raw HTML files, if SAVE_HTML = True)
"""

import time
import random
import sqlite3
import logging
import os
from urllib.parse import urljoin, urlparse, urlencode
from datetime import datetime

import cloudscraper                    # handles Cloudflare JS challenges
from bs4 import BeautifulSoup

# ─────────────────────────────────────────────────────────────
# CONFIGURATION  (edit these as needed)
# ─────────────────────────────────────────────────────────────
BASE_URL    = "https://looksmax.org"
DB_FILE     = "looksmax_archive.db"
SAVE_HTML   = False          # set True to also dump raw HTML to disk
HTML_DIR    = "pages"
DELAY_MIN   = 2.0            # seconds between requests (be polite)
DELAY_MAX   = 4.0
MAX_RETRIES = 5
LOG_LEVEL   = logging.INFO
# ─────────────────────────────────────────────────────────────

logging.basicConfig(
    level=LOG_LEVEL,
    format="%(asctime)s [%(levelname)s] %(message)s",
```

```
    handlers=[
        logging.FileHandler("archiver.log"),
        logging.StreamHandler()
    ]
)
log = logging.getLogger(__name__)

# ── Database setup ──────────────────────────────────────────────────────────

def init_db(conn):
    conn.executescript("""
        CREATE TABLE IF NOT EXISTS forums (
            id          INTEGER PRIMARY KEY,
            xf_id       TEXT UNIQUE,
            title       TEXT,
            description TEXT,
            url         TEXT,
            parent_id   TEXT,
            scraped_at  TEXT
        );

        CREATE TABLE IF NOT EXISTS threads (
            id          INTEGER PRIMARY KEY,
            xf_id       TEXT UNIQUE,
            forum_id    TEXT,
            title       TEXT,
            author      TEXT,
            post_date   TEXT,
            reply_count TEXT,
            view_count  TEXT,
            url         TEXT,
            scraped_at  TEXT
        );

        CREATE TABLE IF NOT EXISTS posts (
            id          INTEGER PRIMARY KEY,
            xf_id       TEXT UNIQUE,
            thread_id   TEXT,
            author      TEXT,
            post_date   TEXT,
            content     TEXT,
            url         TEXT,
            scraped_at  TEXT
```

```
        );

        CREATE TABLE IF NOT EXISTS scrape_state (
            key   TEXT PRIMARY KEY,
            value TEXT
        );
    """)
    conn.commit()


# ── HTTP helpers ──────────────────────────────────────────────────────────────

def make_scraper():
    """cloudscraper automatically bypasses Cloudflare anti-bot pages."""
    scraper = cloudscraper.create_scraper(
        browser={"browser": "chrome", "platform": "windows", "mobile": False}
    )
    scraper.headers.update({
        "Accept-Language": "en-US,en;q=0.9",
        "Referer": BASE_URL,
    })
    return scraper


def fetch(scraper, url, retries=0):
    """Fetch a URL with retry logic and polite delays."""
    time.sleep(random.uniform(DELAY_MIN, DELAY_MAX))
    try:
        resp = scraper.get(url, timeout=30)
        resp.raise_for_status()
        log.debug(f"GET {url} → {resp.status_code}")
        return resp.text
    except Exception as e:
        if retries < MAX_RETRIES:
            wait = (2 ** retries) * 5
            log.warning(f"Error fetching {url}: {e}. Retrying in {wait}s…")
            time.sleep(wait)
            return fetch(scraper, url, retries + 1)
        else:
            log.error(f"Failed after {MAX_RETRIES} retries: {url}")
            return None
```

```
def soup(html):
    return BeautifulSoup(html, "html.parser")


def save_html(url, html):
    if not SAVE_HTML or not html:
        return
    os.makedirs(HTML_DIR, exist_ok=True)
    path = urlparse(url).path.strip("/").replace("/", "_") or "index"
    filepath = os.path.join(HTML_DIR, path + ".html")
    with open(filepath, "w", encoding="utf-8") as f:
        f.write(html)


# ── State helpers (resume support) ──────────────────────────────────────────

def get_state(conn, key):
    row = conn.execute("SELECT value FROM scrape_state WHERE key=?", (key,)).fetchone()
    return row[0] if row else None


def set_state(conn, key, value):
    conn.execute("INSERT OR REPLACE INTO scrape_state VALUES (?,?)", (key, str(value)))
    conn.commit()


# ── XenForo parsers ─────────────────────────────────────────────────────────

def parse_forum_list(html):
    """
    Extract sub-forums from a category/forum page.
    Returns list of dicts: {xf_id, title, description, url}
    """
    s = soup(html)
    forums = []
    for node in s.select("div.node--forum, div[class*='node--forum']"):
        link = node.select_one("a.node-title, h3.node-title a")
        if not link:
            continue
        href = urljoin(BASE_URL, link["href"])
        # XenForo forum URLs: /forums/title.123/
```

```
        xf_id = href.rstrip("/").split(".")[-1]
        desc_el = node.select_one("div.node-description")
        forums.append({
            "xf_id":       xf_id,
            "title":       link.get_text(strip=True),
            "description": desc_el.get_text(strip=True) if desc_el else "",
            "url":         href,
        })
    return forums


def parse_thread_list(html):
    """
    Extract thread stubs from a forum listing page.
    Returns list of dicts + next_page URL or None.
    """
    s = soup(html)
    threads = []
    for row in s.select("div.structItem--thread"):
        title_el = row.select_one("div.structItem-title a[data-tp-primary]")
        if not title_el:
            title_el = row.select_one("div.structItem-title a")
        if not title_el:
            continue
        href = urljoin(BASE_URL, title_el["href"])
        xf_id = href.rstrip("/").split(".")[-1]

        author_el = row.select_one("a.username")
        date_el   = row.select_one("time")
        replies_el = row.select_one("dd.pairs--justified dl dd")  # reply count
        views_el  = row.select_one("dl.pairs--justified:last-child dd")

        threads.append({
            "xf_id":       xf_id,
            "title":       title_el.get_text(strip=True),
            "author":      author_el.get_text(strip=True) if author_el else "",
            "post_date":   date_el["datetime"] if date_el and date_el.has_attr("datetime") else "",
            "reply_count": replies_el.get_text(strip=True) if replies_el else "",
            "view_count":  views_el.get_text(strip=True)  if views_el  else "",
            "url":         href,
        })

    # Next page
    next_link = s.select_one("a[rel='next'], .pageNav-jump--next")
```

```
    next_url  = urljoin(BASE_URL, next_link["href"]) if next_link and next_link.has_attr("href") else
None
    return threads, next_url


def parse_posts(html, thread_url):
    """
    Extract all posts from a thread page.
    Returns list of dicts + next_page URL or None.
    """
    s = soup(html)
    posts = []
    for article in s.select("article.message--post, article[class*='message']"):
        xf_id     = article.get("data-content", "").replace("post-", "")
        author_el = article.select_one("h4.message-name a, span.username")
        date_el   = article.select_one("time.u-dt")
        body_el   = article.select_one("div.message-body div.bbWrapper,
div.message-userContent")

        posts.append({
            "xf_id":     xf_id or article.get("id", ""),
            "author":    author_el.get_text(strip=True) if author_el else "",
            "post_date": date_el["datetime"] if date_el and date_el.has_attr("datetime") else "",
            "content":   body_el.get_text(separator="\n", strip=True) if body_el else "",
            "url":       thread_url,
        })

    next_link = s.select_one("a[rel='next'], .pageNav-jump--next")
    next_url  = urljoin(BASE_URL, next_link["href"]) if next_link and next_link.has_attr("href") else
None
    return posts, next_url


# ── Main crawl logic
─────────────────────────────────────────────────────────────────────


def scrape_thread(scraper, conn, thread):
    """Scrape all pages of a single thread and save posts."""
    thread_id  = thread["xf_id"]
    url        = thread["url"]
    page_num   = 1
    now        = datetime.utcnow().isoformat()

    # Resume: find the last page we got to
```

```
    last_page = get_state(conn, f"thread_{thread_id}_last_page")
    if last_page:
        page_num = int(last_page) + 1
        url = thread["url"].rstrip("/") + f"/page-{page_num}"
        log.info(f"  Resuming thread {thread_id} from page {page_num}")

    while url:
        log.info(f"  Thread {thread_id} page {page_num}: {url}")
        html = fetch(scraper, url)
        if not html:
            break
        save_html(url, html)

        posts, next_url = parse_posts(html, thread["url"])
        for p in posts:
            try:
                conn.execute("""
                    INSERT OR IGNORE INTO posts
                        (xf_id, thread_id, author, post_date, content, url, scraped_at)
                    VALUES (?,?,?,?,?,?,?)
                """, (p["xf_id"], thread_id, p["author"], p["post_date"],
                      p["content"], p["url"], now))
            except sqlite3.IntegrityError:
                pass
        conn.commit()

        set_state(conn, f"thread_{thread_id}_last_page", page_num)
        page_num += 1
        url = next_url

    set_state(conn, f"thread_{thread_id}_done", "1")


def scrape_forum(scraper, conn, forum):
    """Scrape all thread listings in a forum, then scrape each thread."""
    forum_id = forum["xf_id"]
    url      = forum["url"]
    page_num = 1
    now      = datetime.utcnow().isoformat()

    # Resume thread list
    last_page = get_state(conn, f"forum_{forum_id}_last_page")
    if last_page:
        page_num = int(last_page) + 1
```

```
    url = forum["url"].rstrip("/") + f"/page-{page_num}"
    log.info(f"Resuming forum {forum_id} thread list from page {page_num}")

log.info(f"Forum: {forum['title']} ({forum_id})")

# 1. Collect all thread stubs
while url:
    log.info(f"  Listing page {page_num}: {url}")
    html = fetch(scraper, url)
    if not html:
        break
    save_html(url, html)

    threads, next_url = parse_thread_list(html)
    for t in threads:
        try:
            conn.execute("""
                INSERT OR IGNORE INTO threads
                    (xf_id, forum_id, title, author, post_date,
                     reply_count, view_count, url, scraped_at)
                VALUES (?,?,?,?,?,?,?,?,?)
            """, (t["xf_id"], forum_id, t["title"], t["author"],
                  t["post_date"], t["reply_count"], t["view_count"],
                  t["url"], now))
        except sqlite3.IntegrityError:
            pass
    conn.commit()
    set_state(conn, f"forum_{forum_id}_last_page", page_num)
    page_num += 1
    url = next_url

set_state(conn, f"forum_{forum_id}_threads_listed", "1")

# 2. Scrape each thread
threads = conn.execute(
    "SELECT xf_id, url FROM threads WHERE forum_id=?", (forum_id,)
).fetchall()

for row in threads:
    t_id, t_url = row
    if get_state(conn, f"thread_{t_id}_done"):
        log.debug(f"  Skipping already-done thread {t_id}")
        continue
    scrape_thread(scraper, conn, {"xf_id": t_id, "url": t_url})
```

```
    set_state(conn, f"forum_{forum_id}_done", "1")


def discover_forums(scraper, conn):
    """Walk the homepage and find all forum nodes recursively."""
    log.info("Discovering forums from homepage…")
    html = fetch(scraper, BASE_URL)
    if not html:
        log.error("Could not fetch homepage.")
        return []

    save_html(BASE_URL, html)
    s = soup(html)
    now = datetime.utcnow().isoformat()

    found = []
    # XenForo homepage lists all nodes in .p-body-pageContent
    for node in s.select("div.node--forum"):
        link = node.select_one("a.node-title, h3.node-title a")
        if not link:
            continue
        href = urljoin(BASE_URL, link["href"])
        xf_id = href.rstrip("/").split(".")[-1]
        desc_el = node.select_one("div.node-description")
        forum = {
            "xf_id":       xf_id,
            "title":       link.get_text(strip=True),
            "description": desc_el.get_text(strip=True) if desc_el else "",
            "url":         href,
            "parent_id":   None,
        }
        conn.execute("""
            INSERT OR IGNORE INTO forums
                (xf_id, title, description, url, parent_id, scraped_at)
            VALUES (?,?,?,?,?,?)
        """, (forum["xf_id"], forum["title"], forum["description"],
              forum["url"], forum["parent_id"], now))
        found.append(forum)

    conn.commit()
    log.info(f"Found {len(found)} top-level forum nodes.")
    return found
```

```
def main():
    conn = sqlite3.connect(DB_FILE)
    init_db(conn)
    scraper = make_scraper()

    # ── Step 1: discover forums ──
    if not get_state(conn, "forums_discovered"):
        forums = discover_forums(scraper, conn)
        set_state(conn, "forums_discovered", "1")
    else:
        log.info("Forums already discovered, loading from DB…")
        rows = conn.execute("SELECT xf_id, title, description, url, parent_id FROM
forums").fetchall()
        forums = [{"xf_id": r[0], "title": r[1], "description": r[2],
                   "url": r[3], "parent_id": r[4]} for r in rows]

    # ── Step 2: scrape each forum ──
    for forum in forums:
        if get_state(conn, f"forum_{forum['xf_id']}_done"):
            log.info(f"Skipping already-done forum: {forum['title']}")
            continue
        scrape_forum(scraper, conn, forum)

    # ── Done ──
    post_count   = conn.execute("SELECT COUNT(*) FROM posts").fetchone()[0]
    thread_count = conn.execute("SELECT COUNT(*) FROM threads").fetchone()[0]
    forum_count  = conn.execute("SELECT COUNT(*) FROM forums").fetchone()[0]

    log.info("=" * 50)
    log.info("ARCHIVE COMPLETE")
    log.info(f"  Forums:  {forum_count}")
    log.info(f"  Threads: {thread_count}")
    log.info(f"  Posts:   {post_count}")
    log.info(f"  Database: {DB_FILE}")
    log.info("=" * 50)

    conn.close()


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

Querying the database afterwards

Run this to browse:

```
import sqlite3
conn = sqlite3.connect("looksmax_archive.db")

# Search all posts for a keyword
results = conn.execute(
    "SELECT author, content FROM posts WHERE content LIKE '%looksmaxxing%'"
).fetchall()

# Get all threads in a forum
threads = conn.execute(
    "SELECT title, reply_count FROM threads WHERE forum_id='123'"
).fetchall()
```