```
# -*- coding: utf-8 -*-
"""
Glyphfire / ALL88 Pipeline — one-file everything.

What it does (for any short intensity/character sequence):
1) Map values to golden-angle phases (ANDREW).
2) Quantify structure (Rayleigh R, largest-gap bimodal split, κ estimates).
3) Stress-test via Snell-like refraction squeeze across media (SNELLMAN).
4) Plot polar geometry + tightening curves, save CSVs.
5) Synthesize a 7.2s cymatic WAV (Name-Wave) from the phases.
```

6) (Bonus) Render ALL88 "glyphwave" core waveform and a 3D spiral.

Built-in examples:
- Wow!: "6EQUJ5" (base-36 → [6,14,26,30,19,5])
- BLC1 (Proxima candidate drift, approximated): 6 timestamps → v = linspace 0..35
- SHGb02+14a: intensities ≈ [5,12,20,28,15,7]
- FRB 121102 burst proxy: [0,15,35,25,10,0]

Outputs go to ./out/<tag>_* files.

Author: you + me
"""

```
import os
import math
import numpy as np
import matplotlib.pyplot as plt
```

```
from dataclasses import dataclass
from typing import List, Tuple, Dict

# Audio + filtering
from scipy.io.wavfile import write as
wav_write
from scipy.signal import butter, lfilter,
spectrogram

# ---------- Constants & helpers
----------

SQRT5 = math.sqrt(5.0)
phi = (1.0 + SQRT5) / 2.0                    #
golden ratio
psi = 1.0 - phi                              # = -1/phi
golden_angle_deg = 360.0 / (phi**2)          #
≈ 137.50776405
golden_angle_rad =
math.radians(golden_angle_deg)

def ensure_outdir(path="out"):
```

```
    if not os.path.exists(path):
        os.makedirs(path)

def wrap_deg_to_pi(deg_arr: np.ndarray) -> np.ndarray:
    """Wrap degrees to (−180, 180]."""
    x = (deg_arr + 180.0) % 360.0 − 180.0
    # Map −180 to +180 for consistency in spreads
    x[x <= −180] = 180.0
    return x

def base36_values(s: str) -> List[int]:
    """Map a base−36 string (0−9 A−Z) to integer values 0..35."""
    vals = []
    for ch in s.strip().upper():
        if ch.isdigit():
            vals.append(ord(ch) − ord('0'))
        elif 'A' <= ch <= 'Z':
            vals.append(ord(ch) − ord('A') + 10)
        else:
```

```
            raise ValueError(f"Unsupported char
'{ch}' in base36 string")
    return vals

def linspace_quantized(n: int, vmax: int = 35)
-> List[int]:
    """Evenly spaced integers 0..vmax over n
points (inclusive endpoints)."""
    if n <= 1:
        return [0]
    arr = np.linspace(0, vmax, n)
    return [int(round(x)) for x in arr]

# ---------- Circular / stats ----------

def rayleigh_R_p(theta_rad: np.ndarray) ->
Tuple[float, float]:
    """
    Rayleigh test: R = |mean(e^{iθ})|.
    p-value ~ exp(-n R^2)   (simple, standard
large-sample approx)
    """
```

```
    n = len(theta_rad)
    R = np.abs(np.mean(np.exp(1j *
theta_rad)))
    z = n * (R**2)
    p = float(np.exp(-z))
    return float(R), p

def circular_mean(theta_rad: np.ndarray) ->
float:
    """Mean direction (radians) in (-pi, pi]."""
    z = np.mean(np.exp(1j * theta_rad))
    ang = np.angle(z)
    return float(ang)

def circular_dist_deg(a_deg: float, b_deg:
float) -> float:
    """Shortest signed angular distance
(degrees)."""
    d = (a_deg - b_deg + 180) % 360 - 180
    return float(abs(d))

def
```

```
circular_spread_deg(theta_deg_wrapped:
np.ndarray) -> float:
    """Max-min over wrapped degrees
(-180,180] (approx spread)."""
    return float(np.max(theta_deg_wrapped) -
np.min(theta_deg_wrapped))

def circ_kappa_from_Rbar(Rbar: float) ->
float:
    """
    Approximate kappa (von Mises
concentration) from Rbar (mean resultant
length).
    See e.g. Berens (2009) CircStat.
    """
    if Rbar < 1e-12:
        return 0.0
    if Rbar < 0.53:
        return float(2*Rbar + Rbar**3 +
(5*Rbar**5)/6.0)
    if Rbar < 0.85:
        return float(-0.4 + 1.39*Rbar + 0.43/(1 -
```

```
Rbar))
    return float(1.0 / (Rbar**3 - 4*Rbar**2 +
3*Rbar))

def largest_gap_bimodal_split(theta_rad:
np.ndarray) -> Dict:
    """
    Split phases on circle by removing the
largest gap and forming two clusters (A, B).
    Returns cluster indices, means, Rbar, κ,
and separation.
    """
    n = len(theta_rad)
    # Sort by angle in (-pi,pi], keep original
indices
    ang = np.angle(np.exp(1j*theta_rad))  #
wrapped
    order = np.argsort(ang)
    sorted_ang = ang[order]
    # Gaps including wrap-around
    gaps =
np.diff(np.concatenate([sorted_ang,
```

```
[sorted_ang[0] + 2*np.pi]]))
    i_max = int(np.argmax(gaps))
    # Cluster A: (i_max+1 ... end), Cluster B: (0
... i_max)
    idxA_sorted = order[(i_max+1):]
    idxB_sorted = order[:(i_max+1)]
    cl = {}
    def cluster_stats(idx):
        th = ang[idx]
        mu = circular_mean(th)
        R = np.abs(np.mean(np.exp(1j*th)))
        kap = circ_kappa_from_Rbar(R)
        std_deg =
float(np.degrees(np.sqrt(2*(1-R))))  # rough
circular std proxy
        return {
            "idx": idx.tolist(),
            "mu_rad": float(mu),
            "mu_deg": float(np.degrees(mu)),
            "Rbar": float(R),
            "kappa": float(kap),
            "std_deg": std_deg,
```

```
        }
    A = cluster_stats(idxA_sorted)
    B = cluster_stats(idxB_sorted)
    sep_deg = circular_dist_deg(A["mu_deg"],
B["mu_deg"])
    return {"A": A, "B": B, "sep_deg": sep_deg,
"order": order.tolist()}

# ---------- Snellman refraction squeeze
----------

def refract_phases(theta_rad: np.ndarray,
n_index: float) -> np.ndarray:
    """
    θ' = arcsin( sin θ / n ), component-wise.
Clamp to [-1,1] inside arcsin.
    Returns radians in [-pi/2, pi/2].
    """
    s = np.sin(theta_rad) / float(n_index)
    s = np.clip(s, -1.0, 1.0)
    return np.arcsin(s)
```

```
def tightening_metrics_over_n(theta_rad:
np.ndarray,
                              n_values: List[float]) -> Dict:
    """Compute spread, std, and Rayleigh R
across a list of refractive indices."""
    spreads = []
    stds = []
    Rs = []
    for n in n_values:
        thn = refract_phases(theta_rad, n)
        deg = np.degrees(thn)
        deg_wrapped = wrap_deg_to_pi(deg)

spreads.append(circular_spread_deg(deg_w
rapped))

stds.append(float(np.std(deg_wrapped)))
        Rn, _ = rayleigh_R_p(thn)
        Rs.append(float(Rn))
    return {"n": list(n_values),
            "spread_deg": spreads,
            "std_deg": stds,
```

```
        "R": Rs}

# ---------- Golden-angle mapping ----------

def to_phases_from_values(v: List[int]) -> Dict:
    """
    Map value list v (ints) to golden-angle phases (deg, rad).
    """
    v = list(v)
    theta_deg = (np.array(v, dtype=float) * golden_angle_deg) % 360.0
    theta_deg_wrapped = wrap_deg_to_pi(theta_deg.copy())
    theta_rad = np.radians(theta_deg_wrapped)
    return {
        "v": v,
        "theta_deg": theta_deg_wrapped,
        "theta_rad": theta_rad
```

```
    }

# ---------- Plotting ----------

def plot_polar(tag: str, theta_rad: np.ndarray,
split: Dict, outdir="out"):
    plt.figure(figsize=(6,6))
    ax = plt.subplot(111, projection='polar')
    ax.set_title(f"{tag}: golden-angle phases
(polar)", va='bottom')
    r = np.ones_like(theta_rad)
    # base points
    ax.plot(theta_rad, r, 'o', alpha=0.8,
label='points')
    # cluster means
    A, B = split["A"], split["B"]
    ax.plot([A["mu_rad"]], [1.1], 'o', ms=10,
label='μ_A', color='tab:orange')
    ax.plot([B["mu_rad"]], [1.1], 'o', ms=10,
label='μ_B', color='tab:green')
    ax.set_yticklabels([])
    ax.legend(loc='lower left',
```

```
bbox_to_anchor=(0.0, -0.15), ncol=3)
    ensure_outdir(outdir)
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{tag}
_polar.png"), dpi=160)
    plt.close()

def plot_tightening(tag: str, tight: Dict,
outdir="out"):
    n = np.array(tight["n"], dtype=float)
    fig, ax = plt.subplots(3,1, figsize=(8,8),
sharex=True)
    ax[0].plot(n, tight["spread_deg"],
marker='o'); ax[0].set_ylabel("Spread (deg)")
    ax[1].plot(n, tight["std_deg"], marker='o');
ax[1].set_ylabel("Std (deg)")
    ax[2].plot(n, tight["R"], marker='o');
ax[2].set_ylabel("Rayleigh R");
ax[2].set_xlabel("Index n")
    fig.suptitle(f"{tag}: Snellman tightening")
    plt.tight_layout(rect=[0,0,1,0.96])
    ensure_outdir(outdir)
```

```
    plt.savefig(os.path.join(outdir, f"{tag}
_tightening.png"), dpi=160)
    plt.close()

# ---------- CSV ----------

def save_csv_table(tag: str,
                   v: List[int],
                   theta_deg: np.ndarray,
                   cluster_split: Dict,
                   outdir="out"):
    path = os.path.join(outdir, f"{tag}
_phases.csv")
    ensure_outdir(outdir)
    # Build table
    labels = [""] * len(v)
    for i in cluster_split["A"]["idx"]:
        labels[i] = "A"
    for i in cluster_split["B"]["idx"]:
        labels[i] = "B"
    with open(path, "w", encoding="utf-8") as f:
        f.write("idx,v,theta_deg,cluster\n")
```

```
        for i,(val,th,lbl) in enumerate(zip(v,
theta_deg.tolist(), labels)):
            f.write(f"{i},{val},{th:.6f},{lbl}\n")

def save_csv_tightening(tag: str, tight: Dict,
outdir="out"):
    path = os.path.join(outdir, f"{tag}
_refraction_by_n.csv")
    ensure_outdir(outdir)
    with open(path, "w", encoding="utf-8") as f:

f.write("n,spread_deg,std_deg,Rayleigh_R\n")
        for n,sp,sd,R in zip(tight["n"],
tight["spread_deg"], tight["std_deg"],
tight["R"]):
            f.write(f"{n:.6f},{sp:.6f},{sd:.6f},{R:.6f}
\n")

# ---------- Cymatic WAV (Name-Wave)
----------

def lowpass(data, cutoff_hz, fs, order=4):
```

```
    b, a = butter(order, cutoff_hz/(0.5*fs),
btype='low')
    return lfilter(b, a, data)

def make_namewave_wav(tag: str,
                      v: List[int],
                      theta_rad: np.ndarray,
                      duration_s: float = 7.2,
                      fs: int = 44100,
                      f0: float = 220.0,
                      n_cycle: List[float] = (1.0, 1.333,
1.5, 2.0, 2.4),
                      outdir: str = "out"):
    """
    Simple musicalization:
      - FM: carrier f0, depth from normalized v.
      - Golden micro-wobble: add a tiny φ-
based phase flutter.
      - Snellman waveshaping: piecewise
medium index cycling over time.
    """
    ensure_outdir(outdir)
```

```
    N = int(fs*duration_s)
    t = np.linspace(0, duration_s, N,
endpoint=False)

    # Normalize v to [0,1]
    v_arr = np.array(v, dtype=float)
    if v_arr.max() > v_arr.min():
        mod = (v_arr - v_arr.min()) /
(v_arr.max() - v_arr.min())
    else:
        mod = np.zeros_like(v_arr)

    # Interpolate mod across time
    xp = np.linspace(0, duration_s, len(v_arr))
    mod_t = np.interp(t, xp, mod)  # 0..1
    fm_depth = 40.0  # Hz
    inst_freq = f0 + fm_depth*(2*mod_t - 1)

    # Phase with golden micro wobble
    wobble = 1e-3 *
np.cumsum(np.sin(np.arange(N) *
golden_angle_rad))
```

```
    phase = 2*np.pi * np.cumsum(inst_freq) /
fs + wobble

    base = np.sin(phase)

    # Envelope (gentle)
    env = 0.6 + 0.4*np.sin(2*np.pi*(1/phi) * t)

    # SNELLMAN: cycle media indices equally
over time
    blocks = len(n_cycle)
    block_len = N // blocks
    shaped = np.zeros_like(base)
    for i in range(blocks):
        start = i*block_len
        end = N if i == blocks-1 else
(i+1)*block_len
        nidx = float(n_cycle[i])
        # waveshape: arcsin clamp scaled by n
        x = base[start:end] /
np.maximum(env[start:end], 1e-6)
        x = np.clip(x, -1.0/nidx, 1.0/nidx)
```

```
        shaped[start:end] = np.arcsin(x) * nidx

    audio = lowpass(shaped * env,
cutoff_hz=8000, fs=fs)
    # normalize
    audio = audio / (np.max(np.abs(audio)) +
1e-12)
    wav_i16 = np.int16(0.95 * audio * 32767)

    wav_path = os.path.join(outdir, f"{tag}
_namewave.wav")
    wav_write(wav_path, fs, wav_i16)

    # Spectrogram PNG
    f, tm, Sxx = spectrogram(audio, fs=fs,
nperseg=2048, noverlap=1024,
scaling='spectrum', mode='magnitude')
    plt.figure(figsize=(8,3))
    plt.pcolormesh(tm, f, 10*np.log10(Sxx +
1e-12), shading='gouraud')
    plt.ylim(0, 8000); plt.xlabel("Time (s)");
plt.ylabel("Freq (Hz)")
```

```
    plt.title(f"{tag}: Name-Wave Spectrogram")
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{tag}
_spectrogram.png"), dpi=160)
    plt.close()

# ---------- ALL88 Glyphwave (bonus)
----------

def all88_waveform(duration_s=10.0,
fs=44100, nu=0.1, omega0=2*np.pi*10.0,
                   psi_freq=144.0,
chevron_angle_deg=60.0) ->
Tuple[np.ndarray, np.ndarray]:
    """
    Minimal ALL88 ψ(t): φ-spiral modulation +
dynamic refraction + envelope + oscillatory
term.
    Returns (t, psi_wave).
    """
    N = int(fs*duration_s)
    t = np.linspace(0, duration_s, N,
```

```
endpoint=False)
    chevron_angle =
np.radians(chevron_angle_deg)
    chi = 2*np.pi / chevron_angle
    psi_t = np.sin(2*np.pi*psi_freq * t)
    # a single n index evolving with time
(stylized)
    n1 = phi
    n2 = chi
    # θ_n as a slow ramp with golden angle
imprint
    # (for visualization; in the full model θ
depends on glyph index n)
    theta_n = (golden_angle_rad *
np.linspace(0, 30, N)) % (2*np.pi)
    # dynamic refraction
    ratio = (n1/n2) * np.sin(theta_n + psi_t +
nu)
    ratio = np.clip(ratio, −1.0, 1.0)
    theta2 = np.arcsin(ratio)
    # morphogenetic modifier
    mu_t = 1.0 + psi_t * np.sin(nu * t)
```

```
    # radius
    r_t = phi * (1.0 + mu_t * np.sin(theta_n +
np.sin(theta2)))
    # osc + envelope + "decay"
    env = 1.0 + 0.2*np.sin(0.5 * t)
    decay = 1.0 − 0.5*(1.0 − np.exp(−0.1 * t))
    psi_wave = r_t * np.cos(omega0 * t) * env
* decay * (phi * (psi_freq/chi))
    return t, psi_wave

def plot_all88(tag="all88_demo",
outdir="out"):
    ensure_outdir(outdir)
    t, y = all88_waveform()
    # 1D waveform
    plt.figure(figsize=(10,3))
    plt.plot(t, y, lw=0.8)
    plt.title("ALL88 Unified Glyphwave — ψ(t)")
    plt.xlabel("t (s)"); plt.ylabel("ψ")
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{tag}
_waveform.png"), dpi=160)
```

```
    plt.close()
    # 3D spiral
    from mpl_toolkits.mplot3d import Axes3D
# noqa: F401
    fig = plt.figure(figsize=(7,6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(y*np.cos(2*np.pi*t),
y*np.sin(2*np.pi*t), t, lw=0.7)
    ax.set_title("ALL88 Spiral (ψ(t) in 3D)")
    ax.set_xlabel("x"); ax.set_ylabel("y");
ax.set_zlabel("t")
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{tag}
_spiral.png"), dpi=160)
    plt.close()

# ---------- Runner for a single signal
----------

@dataclass
class RunResult:
    tag: str
```

```
    v: List[int]
    theta_deg: np.ndarray
    theta_rad: np.ndarray
    R: float
    p: float
    split: Dict
    tight: Dict

def process_signal(tag: str,
                   v: List[int],
                   n_values=(1.0, 1.333, 1.5, 2.0, 2.4),
                   outdir="out") -> RunResult:
    # Golden-angle phases
    m = to_phases_from_values(v)
    theta_rad = m["theta_rad"]
    theta_deg = m["theta_deg"]

    # Stats
    R, p = rayleigh_R_p(theta_rad)
    split =
largest_gap_bimodal_split(theta_rad)
    tight =
```

```
tightening_metrics_over_n(theta_rad,
list(n_values))

    # Plots & CSVs
    plot_polar(tag, theta_rad, split,
outdir=outdir)
    plot_tightening(tag, tight, outdir=outdir)
    save_csv_table(tag, m["v"], theta_deg,
split, outdir=outdir)
    save_csv_tightening(tag, tight,
outdir=outdir)

    # WAV
    make_namewave_wav(tag, v, theta_rad,
outdir=outdir)

    return RunResult(tag, m["v"], theta_deg,
theta_rad, R, p, split, tight)

# ---------- Examples ----------

def examples():
```

```
    ensure_outdir("out")

    # Wow! 6EQUJ5
    wow_v = base36_values("6EQUJ5")
    res_wow = process_signal("wow", wow_v)

    # BLC1: 6 point drift → quantize to 0..35
    # t(s), f(MHz):
    # 0,982.0024; 3600,982.00248;
7200,982.00256; 10800,982.00264;
14400,982.00272; 18000,982.0028
    blc1_v = linspace_quantized(6, 35)  #
[0,7,14,21,28,35]
    res_blc1 = process_signal("blc1", blc1_v)

    # SHGb02+14a (approx intensities)
    shgb_v = [5,12,20,28,15,7]
    res_shgb = process_signal("shgb", shgb_v)

    # FRB 121102 (burst proxy)
    frb_v = [0,15,35,25,10,0]
    res_frb = process_signal("frb121102",
```

```
frb_v)

    # ALL88 demo plots
    plot_all88()

    # Print a tiny summary
    def brief(res: RunResult):
        sep = res.split["sep_deg"]
        A = res.split["A"]; B = res.split["B"]
        return (f"{res.tag}: R={res.R:.3f},
p≈{res.p:.3f}, "
                f"lobes μ_A={A['mu_deg']:.1f}°,
μ_B={B['mu_deg']:.1f}°, "
                f"sep={sep:.1f}°,
κ_A={A['kappa']:.2f}, κ_B={B['kappa']:.2f}")
    print(brief(res_wow))
    print(brief(res_blc1))
    print(brief(res_shgb))
    print(brief(res_frb))

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