```
    #!/usr/bin/env python3
"""
Binary Options Signal Bot (Termux/Replit-ready)
    - 3-minute sampling (approximate candles via sampled price)
- 6-minute expiry signals (CALL / PUT)
- Strategies: Triple Confirmation (trend) & Range Bounce (range)
- Telegram notifications (BOT_TOKEN & CHAT_ID via environment)
- CSV logging
- NOTE: For production, replace data source with broker OHLC API.
"""
    import os
import time
import threading
import csv
from datetime import datetime
import requests
import pandas as pd
import numpy as np
from telegram import Bot
    # ——————————
# CONFIG / ENV
# ——————————
BOT_TOKEN = os.getenv("BOT_TOKEN", "YOUR_BOT_TOKEN")
CHAT_ID = os.getenv("CHAT_ID", "YOUR_CHAT_ID") # string or int
    PAIRS = ["EURUSD", "GBPUSD", "EURJPY", "AUDCAD"]
INTERVAL_SECONDS = 180
CANDLE_HISTORY = 200
EXPIRY_MINUTES = 6
SESSION_WINDOWS_UTC = [
(8, 17), # London
(13, 22) # New York
]
    EMA_SHORT = 9
EMA_LONG = 21
EMA_SLOW = 50
RSI_PERIOD = 14
RSI_UPPER = 70
RSI_LOWER = 30
TRIPLE_RSI_CONFIRM_BUY = 55
TRIPLE_RSI_CONFIRM_SELL = 45
    BOLL_PERIOD = 20
BOLL_STD = 2
STOCH_K = 14
STOCH_D = 3
    LOG_FILE = "signals_log.csv"
    # ——————————
```

```
# Telegram setup
# ——————————
bot = Bot(token=BOT_TOKEN)
    def send_telegram(message: str):
try:
bot.send_message(chat_id=CHAT_ID, text=message)
except Exception as e:
print(f"[Telegram] send error: {e}")
    # ——————————
# Logging
# ——————————
if not os.path.exists(LOG_FILE):
with open(LOG_FILE, "w", newline="") as f:
w = csv.writer(f)
w.writerow(["timestamp_utc", "pair", "signal", "strategy", "price", "expiry_min",
"note"])
    def log_signal(timestamp, pair, signal, strategy, price, expiry, note=""):
with open(LOG_FILE, "a", newline="") as f:
w = csv.writer(f)
w.writerow([timestamp, pair, signal, strategy, f"{price:.6f}" if isinstance(price,
float) else price, expiry, note])
    # ——————————
# Time/session helpers
# ——————————
def in_trading_session(now_utc=None):
if now_utc is None:
now_utc = datetime.utcnow()
h = now_utc.hour
for start, end in SESSION_WINDOWS_UTC:
if start <= h < end:
return True
return False
    # ——————————
# Data fetching
# ——————————
def fetch_latest_price(pair: str):
base = pair[:3]
quote = pair[3:]
try:
url = f"https://api.exchangerate.host/latest?base={base}&symbols={quote}"
res = requests.get(url, timeout=10).json()
rate = res.get("rates", {}).get(quote)
if rate is None:
return None
return float(rate)
except Exception as e:
```

```
print(f"[Data] Error fetching {pair}: {e}")
return None
    # ——————————
# Indicators
# ——————————
def ema(series, period):
s = pd.Series(series)
return s.ewm(span=period, adjust=False).mean().iloc[-1]
    def calculate_rsi(prices, period=14):
s = pd.Series(prices)
delta = s.diff().dropna()
up = delta.clip(lower=0)
down = -1 * delta.clip(upper=0)
roll_up = up.ewm(alpha=1/period, adjust=False).mean()
roll_down = down.ewm(alpha=1/period, adjust=False).mean()
rs = roll_up / roll_down
rsi = 100 - (100 / (1 + rs))
return float(rsi.iloc[-1])
    def bollinger_bands(prices, period=20, stddev=2):
s = pd.Series(prices)
sma = s.rolling(window=period).mean().iloc[-1]
std = s.rolling(window=period).std().iloc[-1]
upper = sma + stddev * std
lower = sma - stddev * std
last = s.iloc[-1]
return upper, lower, last
    def stochastic_k_d(prices, k_period=14, d_period=3):
s = pd.Series(prices)
low_min = s.rolling(window=k_period).min()
high_max = s.rolling(window=k_period).max()
k = 100 * (s - low_min) / (high_max - low_min)
d = k.rolling(window=d_period).mean()
return float(k.iloc[-1]), float(d.iloc[-1])
    # ——————————
# Strategies
# ——————————
def detect_triple_confirmation(prices):
if len(prices) < max(EMA_LONG + 5, RSI_PERIOD + 5):
return None
ema9 = pd.Series(prices).ewm(span=EMA_SHORT, adjust=False).mean().iloc[-
1]
ema21 = pd.Series(prices).ewm(span=EMA_LONG, adjust=False).mean().iloc[-
1]
ema9_prev = pd.Series(prices[:-1]).ewm(span=EMA_SHORT, adjust=False).mean().iloc[-
1] if len(prices) > 1 else ema9
ema21_prev = pd.Series(prices[:-1]).ewm(span=EMA_LONG, adjust=False).mean().iloc[-
```

```
1] if len(prices) > 1 else ema21
rsi = calculate_rsi(prices, RSI_PERIOD)
if ema9_prev <= ema21_prev and ema9 > ema21 and rsi >= TRIPLE_RSI_CONFIRM_BUY
and rsi < RSI_UPPER:
return "CALL"
if ema9_prev >= ema21_prev and ema9 < ema21 and rsi <= TRIPLE_RSI_CONFIRM_SELL
and rsi > RSI_LOWER:
return "PUT"
return None
    def detect_range_bounce(prices):
if len(prices) < max(BOLL_PERIOD + 5, STOCH_K + 5, EMA_SLOW +
1):
return None
ema50 = pd.Series(prices).ewm(span=EMA_SLOW, adjust=False).mean()
last_ema50 = ema50.iloc[-1]
prev_mean = np.mean(prices[-10:-5]) if len(prices) >= 10 else np.mean(prices[:-
1]) if len(prices) > 1 else last_ema50
ema50_slope = last_ema50 - prev_mean
price_scale = abs(last_ema50) if last_ema50 != 0 else 1
if abs(ema50_slope) / price_scale > 0.0006:
return None
upper, lower, last = bollinger_bands(prices, BOLL_PERIOD, BOLL_STD)
k, d = stochastic_k_d(prices, STOCH_K, STOCH_D)
s = pd.Series(prices)
prev_k = 100 * (s.iloc[-2] - s.rolling(window=STOCH_K).min().iloc[-2]) / (s.rolling(window=STOCH_K).max
2] - s.rolling(window=STOCH_K).min().iloc[-2]) if len(prices) > 2 else k
if last <= lower and (k > d) and (k < 20):
return "CALL"
if last >= upper and (k < d) and (k > 80):
return "PUT"
return None
    # ——————————
# Worker thread
# ——————————
class PairWorker(threading.Thread):
def ___init___(self, pair):
super().___init___(daemon=True)
self.pair = pair
self.price_history = []
self.last_sent = None
    def run(self):
print(f"[{self.pair}] Worker started.")
while True:
try:
now = datetime.utcnow()
if in_trading_session(now):
```

```
price = fetch_latest_price(self.pair)
if price is not None:
self.price_history.append(price)
if len(self.price_history) > CANDLE_HISTORY:
self.price_history.pop(0)
print(f"[{self.pair}] {now.strftime('%Y-%m-%d %H:%M:%S')} price={price:.6f}
samples={len(self.price_history)}")
signal = None
strategy = None
if len(self.price_history) >= max(BOLL_PERIOD, EMA_LONG, RSI_PERIOD,
STOCH_K):
ema50_series = pd.Series(self.price_history).ewm(span=EMA_SLOW, adjust=False).mean()
ema50_value = float(ema50_series.iloc[-1])
ema50_prev_mean = float(pd.Series(self.price_history[-10:-5]).mean()) if len(self.price_history)
>= 10 else ema50_value
ema50_slope = abs(ema50_value - ema50_prev_mean)
if ema50_slope / (abs
```