""" DARKBOT™: Resonant Field Intelligence Architecture

Configuration module for the DARKBOT™ system.

This file contains the configuration classes and utilities for the DARKBOT™ Resonant Field Intelligence Architecture.

from dataclasses import dataclass, asdict, field from typing import List, Dict, Any, Optional, Tuple import yaml from pathlib import Path

@dataclass class FieldConfig: """ Configuration for field parameters.

```
Attributes:
    dimensions: Number of dimensions in the field
    phi: Golden ratio value for temporal alignment
    epsilon: Small constant for numerical stability
"""
dimensions: int = 512
phi: float = 1.618033988749895  # Golden ratio
epsilon: float = 1e-8

def __post_init__(self):
    """Validate configuration parameters."""
    if not (64 <= self.dimensions <= 4096):
        raise ValueError(f"Field dimensions {self.dimensions} out of resonant range [64,
4096]")
```

@dataclass class ResonanceConfig: """ Configuration for resonance parameters.

```
    Attributes:
        gamma: Memory decay constant
        chi_high: High coherence threshold
        chi_low: Low coherence threshold
        recursion_depth: Depth of recursive loops
    """
    gamma: float = 0.7  # Memory decay constant
    chi_high: float = 0.85  # High coherence threshold
    chi_low: float = 0.65  # Low coherence threshold
    recursion_depth: int = 3  # Depth of recursive loops

    def __post_init__(self):
        """Validate configuration parameters."""
        if not (0.5 <= self.gamma <= 0.95):
            raise ValueError(f"γ={self.gamma} out of resonant range [0.5, 0.95]")
        if not (0.7 <= self.chi_high <= 0.95):
            raise ValueError(f"χ_high={self.chi_high} out of resonant range [0.7, 0.95]")
        if not (0.5 <= self.chi_low <= 0.7):
            raise ValueError(f"χ_low={self.chi_low} out of resonant range [0.5, 0.7]")
        if not (self.chi_low < self.chi_high):
            raise ValueError(f"Coherence threshold inversion: χ_low={self.chi_low} ≥ χ_high=
    {self.chi_high}")
        if not (1 <= self.recursion_depth <= 7):
            raise ValueError(f"Recursion depth {self.recursion_depth} out of range [1, 7]")
```

@dataclass class NumerologyConfig: """ Configuration for numerological components.

```
Attributes:
    NUM_369: Generative Field Dynamics values
    NUM_157: Identity/Temporal Harmonics values
    NUM_248: Coupling Topology values
    WEIGHTS_369: Weights for 369 components
    WEIGHTS_157: Weights for 157 components
    WEIGHTS_248: Weights for 248 components
"""
# Numerological sequences
_NUM_369: Tuple[int, int, int] = (3, 6, 9)
_NUM_157: Tuple[int, int, int] = (1, 5, 7)
_NUM_248: Tuple[int, int, int] = (2, 4, 8)

# Base weights before normalization
_BASE_WEIGHTS_369: Tuple[float, float, float] = (3, 3, 4)
_BASE_WEIGHTS_157: Tuple[float, float, float] = (2, 3, 5)
_BASE_WEIGHTS_248: Tuple[float, float, float] = (2, 3, 5)

# Normalized weights (calculated in __post_init__)
WEIGHTS_369: List[float] = field(default_factory=list)
WEIGHTS_157: List[float] = field(default_factory=list)
WEIGHTS_248: List[float] = field(default_factory=list)

def __post_init__(self):
    """Calculate normalized weights."""
    self.WEIGHTS_369 = self._normalize_weights(self._BASE_WEIGHTS_369)
    self.WEIGHTS_157 = self._normalize_weights(self._BASE_WEIGHTS_157)
    self.WEIGHTS_248 = self._normalize_weights(self._BASE_WEIGHTS_248)

def _normalize_weights(self, weights):
    """Normalize weights to sum to 1.0."""
    total = sum(weights)
    return [w / total for w in weights]

@property
def NUM_369(self):
    """Get 369 sequence values."""
    return self._NUM_369

@property
def NUM_157(self):
    """Get 157 sequence values."""
    return self._NUM_157

@property
def NUM_248(self):
```

```
    """Get 248 sequence values."""
    return self._NUM_248
```

@dataclass class PerformanceConfig: """ Configuration for performance parameters.

```
Attributes:
    batch_size: Default batch size for processing
    hierarchical_threshold: Dimension threshold for hierarchical processing
"""
batch_size: int = 64
hierarchical_threshold: int = 1024

def __post_init__(self):
    """Validate configuration parameters."""
    if not (1 <= self.batch_size <= 256):
        raise ValueError(f"Batch size {self.batch_size} out of reasonable range [1, 256]")
```

@dataclass class DarkBotConfig: """ Master configuration for the DARKBOT™ system.

```
    Attributes:
        field: Field configuration
        resonance: Resonance configuration
        numerology: Numerology configuration
        performance: Performance configuration
    """
    field: FieldConfig = field(default_factory=FieldConfig)
    resonance: ResonanceConfig = field(default_factory=ResonanceConfig)
    numerology: NumerologyConfig = field(default_factory=NumerologyConfig)
    performance: PerformanceConfig = field(default_factory=PerformanceConfig)

    def to_dict(self) -> Dict[str, Any]:
        """
        Convert configuration to dictionary for serialization.

        Returns:
            Dictionary representation of configuration
        """
        return {
            "field": asdict(self.field),
            "resonance": asdict(self.resonance),
            "numerology": {
                "NUM_369": self.numerology.NUM_369,
                "NUM_157": self.numerology.NUM_157,
                "NUM_248": self.numerology.NUM_248,
                "WEIGHTS_369": self.numerology.WEIGHTS_369,
                "WEIGHTS_157": self.numerology.WEIGHTS_157,
                "WEIGHTS_248": self.numerology.WEIGHTS_248
            },
            "performance": asdict(self.performance)
        }

    @classmethod
    def from_dict(cls, config_dict: Dict[str, Any]) -> 'DarkBotConfig':
        """
        Create configuration from dictionary.

        Args:
            config_dict: Dictionary of configuration values

        Returns:
            DarkBotConfig instance
        """
        field_config = FieldConfig(**config_dict.get("field", {}))
        resonance_config = ResonanceConfig(**config_dict.get("resonance", {}))
```

```
    # Handle numerology specially due to properties
    numerology_dict = config_dict.get("numerology", {})
    numerology_config = NumerologyConfig()

    # Update base values if provided
    numerology_attrs = {
        "NUM_369": "_NUM_369",
        "NUM_157": "_NUM_157",
        "NUM_248": "_NUM_248",
        "WEIGHTS_369": "_BASE_WEIGHTS_369",
        "WEIGHTS_157": "_BASE_WEIGHTS_157",
        "WEIGHTS_248": "_BASE_WEIGHTS_248"
    }

    for public_attr, private_attr in numerology_attrs.items():
        if public_attr in numerology_dict:
            setattr(numerology_config, private_attr, tuple(numerology_dict[public_attr]))

    # Recalculate weights
    numerology_config.__post_init__()

    performance_config = PerformanceConfig(**config_dict.get("performance", {}))

    return cls(
        field=field_config,
        resonance=resonance_config,
        numerology=numerology_config,
        performance=performance_config
    )

@classmethod
def from_yaml(cls, yaml_path: str) -> 'DarkBotConfig':
    """
    Load configuration from YAML file.

    Args:
        yaml_path: Path to YAML configuration file

    Returns:
        DarkBotConfig instance
    """
    with open(yaml_path, 'r') as f:
        config_dict = yaml.safe_load(f)
    return cls.from_dict(config_dict)

def to_yaml(self, yaml_path: str):
    """
```

```
    Save configuration to YAML file.

    Args:
        yaml_path: Path to save YAML configuration
    """
    with open(yaml_path, 'w') as f:
        yaml.dump(self.to_dict(), f, default_flow_style=False)

@classmethod
def load_default(cls) -> 'DarkBotConfig':
    """
    Load default configuration from package data.

    Returns:
        DarkBotConfig instance with default values
    """
    default_path = Path(__file__).parent / 'config' / 'default_config.yaml'
    if default_path.exists():
        return cls.from_yaml(str(default_path))
    return cls()  # Return default instance if file doesn't exist
```

def create_default_config_file(filepath: str = 'darkbot_config.yaml'): """ Create a default configuration file.

```
Args:
    filepath: Path to save the default configuration
"""
config = DarkBotConfig()
config.to_yaml(filepath)
print(f"Default configuration saved to {filepath}")
```

if **name** == "**main**": # Example usage create_default_config_file()