BURSA TEKNİK ÜNİVERSİTESİ

Gerçek Zamanlı Söz Dizimi Vurgulayıcı Projesi Raporu

Nazmi Cirim – 21360859069

Python Söz Dizimi Vurgulayıcı - Teknik Dokümantasyon

Proje Genel Bakışı

Bu belge, BLM0238 Programlama Dilleri Projesi - sözcüksel analiz, ayrıştırma ve Tkinter tabanlı GUI ile söz dizimi vurgulamayı uygulayan gerçek zamanlı bir Python söz dizimi vurgulayıcı için kapsamlı teknik raporu içermektedir.

1. Dil ve Dilbilgisi Seçimi

1.1 Hedef Dil Seçimi

- Seçilen Dil: Python
- Python Seçiminin Gerekçesi:
    - Eğitim Değeri: Python'ın temiz söz dizimi, dil işleme kavramlarını daha erişilebilir kılar.
    - Karmaşıklık Dengesi: Temel kavramları ezici bir karmaşıklık olmadan göstermek için yeterince zengin.
    - Tanıdık Bağlam: Öğrenciler tanıdık olmayan söz dizimi yerine dil işleme konularına odaklanabilir.
    - İyi Tanımlanmış Dilbilgisi: Python'ın açık, belgelenmiş bir dilbilgisi belirtimi vardır.
    - Endüstri İlgisi: Python akademik ve endüstride yaygın olarak kullanılmaktadır.

1.2 Uygulanan Dilbilgisi Alt Kümesi

- Uygulanan Dilbilgisi Öğeleri:
    - **Sözcüksel Öğeler:**
        - `KEYWORDS` (Anahtar Kelimeler):

```
KEYWORDS = {  # Python anahtar kelimeleri kümesi
    'def', 'class', 'if', 'elif', 'else', 'for', 'while', 'try', 'except',
    'finally', 'with', 'as', 'import', 'from', 'return', 'yield', 'break',
    'continue', 'pass', 'lambda', 'and', 'or', 'not', 'in', 'is', 'True',
    'False', 'None', 'global', 'nonlocal', 'assert', 'del', 'raise'
}
```

        - `OPERATORS` (Operatörler):

```
OPERATORS = {  # Python operatörleri kümesi
    '+', '-', '*', '/', '//', '%', '**', '=', '==', '!=', '<', '>',
    '<=', '>=', '+=', '-=', '*=', '/=', '&', '|', '^', '~', '<<', '>>'
}
```

        - `DELIMITERS` (Sınırlayıcılar):

```
DELIMITERS = {  # Python ayraç karakterleri kümesi
    '(', ')', '[', ']', '{', '}', ',', ':', ';', '.', '@'
}
```

    - **Dilbilgisi Kapsam Kararları:**
        - Tam Anahtar Kelime Kümesi: Doğru sınıflandırma için tüm Python anahtar kelimeleri.
        - Kapsamlı Operatörler: Aritmetik, karşılaştırma, mantıksal ve atama operatörleri.
        - Temel Sınırlayıcılar: Parantezler, köşeli parantezler, küme parantezleri ve noktalama işaretleri.
        - Dize Literalleri: Kaçış dizileri ile tek ve çift tırnaklı dizeler.
        - Sayısal Literaller: Tam sayılar ve ondalık sayılar.
        - Yorumlar: Satır sonuna kadar hash stilinde yorumlar.
- **Dışlanan Dilbilgisi Öğeleri:**
    - Karmaşık Dize Biçimleri: Üç tırnaklı dizeler, f-dizeleri, ham dizeler.
    - Gelişmiş Sayısal: Karmaşık sayılar, bilimsel gösterim, ikili/onaltılık sayılar.
    - Dekoratörler: '@' sembolünün ötesinde gelişmiş dekoratör söz dizimi.
    - Lambda İfadeleri: Tam lambda ayrıştırması (yalnızca anahtar kelime olarak ele alınır).
- **Alt Küme Seçiminin Gerekçesi:**
    - Kapsam Yönetimi: Yaygın Python kod desenlerinin %80'ini kapsar.
    - Eğitim Odağı: Temel dil işleme kavramlarını gösterir.
    - Uygulama Fizibilitesi: Akademik proje için yönetilebilir karmaşıklık.
    - Gösterim Değeri: Gerçek dünya uygulanabilirliğini göstermek için yeterli karmaşıklık.

1.3 Token Sınıflandırma Stratejisi

- **Token Tipi Hiyerarşisi:**

```python
class TokenType(Enum):
    """Token types for Python syntax highlighting"""
    KEYWORD = "KEYWORD"         # Anahtar kelimeler (def, if, else, vb.)
    STRING = "STRING"           # String sabitleri ("merhaba", 'dünya', vb.)
    NUMBER = "NUMBER"           # Sayısal sabitler (123, 3.14, vb.)
    COMMENT = "COMMENT"         # Yorum satırları (# ile başlayan)
    OPERATOR = "OPERATOR"       # Operatörler (+, -, *, /, =, vb.)
    DELIMITER = "DELIMITER"     # Ayraçlar (parantez, iki nokta, nokta, vb.)
    IDENTIFIER = "IDENTIFIER"   # Tanımlayıcılar (değişken/fonksiyon isimleri)
    UNKNOWN = "UNKNOWN"         # Tanımlanamayan karakterler
    WHITESPACE = "WHITESPACE"   # Boşluk karakterleri (analiz için)
    NEWLINE = "NEWLINE"         # Satır sonu karakteri
```

- **Sınıflandırma Gerekçesi:**
    - Semantik Gruplama: Token'lar dilsel fonksiyona göre gruplandırılır.
    - Vurgulama Desteği: Her tip, ayrı bir görsel temsile eşlenir.
    - Hata Tespiti: `UNKNOWN` tipi, sözcüksel hataların belirlenmesine yardımcı olur.
    - İşleme Verimliliği: Açık kategoriler, hedeflenen işlemeyi sağlar.

2. Sözcüksel Analiz Detayları

2.1 Tokenizasyon Metodolojisi

- Yaklaşım: Karakter-karakter sonlu durum makinesi.

- Temel Algoritma:

```python
def tokenize(self) -> List[Token]:
    """Tokenize the input code"""
    self.tokens = []  # Token listesini temizle
    self.position = 0
    self.line = 1
    self.column = 1

    while self.position < len(self.code):  # Kodun sonuna kadar döngü
        self._skip_whitespace()  # Boşlukları atla

        if self.position >= len(self.code):
            break  # Kodun sonuna gelindiyse çık

        current_char = self.code[self.position]  # Şu anki karakter

        if current_char == '\n':  # Satır sonu karakteri
            self._add_token(TokenType.NEWLINE, current_char)
            self.line += 1
            self.column = 1
            self.position += 1
        elif current_char == '#':  # Yorum satırı
            self._tokenize_comment()
        elif current_char in ['"', "'"]:  # String başlangıcı
            self._tokenize_string(current_char)
        elif current_char.isdigit():  # Sayı başlangıcı
            self._tokenize_number()
        elif current_char.isalpha() or current_char == '_':  # Tanımlayıcı veya anahtar kelime
            self._tokenize_identifier()
        elif current_char in self.OPERATORS:  # Operatör
            self._tokenize_operator()
        elif current_char in self.DELIMITERS:  # Ayraç
            self._add_token(TokenType.DELIMITER, current_char)
            self.position += 1
            self.column += 1
        else:  # Tanımlanamayan karakter
            self._add_token(TokenType.UNKNOWN, current_char)
            self.position += 1
            self.column += 1

    self._update_statistics()  # İstatistikleri güncelle
    return self.tokens  # Token listesini döndür
```

2.2 Durum Yönetimi

- **Konum Takibi:**
  - Mutlak Konum: Kaynak kodundaki karakter indeksi.
  - Satır Numarası: Geçerli satır (1 tabanlı).
  - Sütun Numarası: Satır içindeki konum (1 tabanlı).
  - Sekme İşleme: Sekmeler sütun hesaplaması için 4 boşluk sayılır.
- **Durum Değişkenleri:**
  - `self.position = 0` # Geçerli karakter indeksi
  - `self.line = 1` # Geçerli satır numarası
  - `self.column = 1` # Geçerli sütun numarası

2.3 Uzmanlaşmış Token İşleme

- **Dize Tokenizasyonu:**

```python
def _tokenize_string(self, quote_char):
    """Tokenize a string literal"""
    start_pos = self.position  # Stringin başladığı pozisyon
    self.position += 1  # Açılış tırnağını atla
    self.column += 1

    while self.position < len(self.code):
        current_char = self.code[self.position]
        if current_char == quote_char:
            self.position += 1  # Kapanış tırnağını da dahil et
            self.column += 1
            break
        elif current_char == '\\' and self.position + 1 < len(self.code):
            self.position += 2  # Kaçış karakterini atla
            self.column += 2
        else:
            self.position += 1
            self.column += 1

    string_text = self.code[start_pos:self.position]  # String metni
    self._add_token(TokenType.STRING, string_text)
```

    - **Temel Özellikler:**
        - Tırnak Eşleştirme: Tek ve çift tırnak işaretlerini işler.
        - Kaçış Dizileri: Ters eğik çizgi kaçışlarını doğru şekilde işler.
        - Kapanmamış Dize Tespiti: Kapanmamış dizeleri belirler.

- **Sayı Tokenizasyonu:**

```python
def _tokenize_number(self):
    """Tokenize a number"""
    start_pos = self.position  # Sayının başladığı pozisyon

    while (self.position < len(self.code) and
           (self.code[self.position].isdigit() or self.code[self.position] == '.')):
        self.position += 1
        self.column += 1

    number_text = self.code[start_pos:self.position]  # Sayı metni
    self._add_token(TokenType.NUMBER, number_text)
```

    - **Özellikler:**
        - Tam Sayı Desteği: Basamak dizileri.
        - Ondalık Sayı Desteği: Ondalık nokta işleme.
        - Basit Uygulama: Temel sayısal desen tanıma.

- **Tanımlayıcı/Anahtar Kelime İşleme:**

```python
def _tokenize_identifier(self):
    """Tokenize an identifier or keyword"""
    start_pos = self.position  # Tanımlayıcının başladığı pozisyon

    while (self.position < len(self.code) and
           (self.code[self.position].isalnum() or self.code[self.position] == '_')):
        self.position += 1
        self.column += 1

    identifier_text = self.code[start_pos:self.position]  # Tanımlayıcı metni

    if identifier_text in self.KEYWORDS:
        self._add_token(TokenType.KEYWORD, identifier_text)  # Anahtar kelime ise
    else:
        self._add_token(TokenType.IDENTIFIER, identifier_text)  # Değilse tanımlayıcı
```

- **Sınıflandırma Mantığı:**
  - Desen Tanıma: Alfasayısal + alt çizgi dizileri.
  - Anahtar Kelime Arama: Sınıflandırma için O(1) hash tablosu araması.
  - Varsayılan Tanımlayıcıya Ayarlama: Anahtar kelime olmayanlar tanımlayıcı olur.

2.4 Sözcüksel Analizde Hata İşleme

- **Hata Tespiti:**
  - Bilinmeyen Karakterler: Herhangi bir token deseniyle eşleşmeyen karakterler.
  - Kapanmamış Dizeler: Kapanış tırnak işareti eksik dizeler.
  - Geçersiz Diziler: Hatalı biçimlendirilmiş sayısal veya tanımlayıcı desenler.
- **Hata Kurtarma:**
  - İşlemeye Devam Et: İlk hatada durma.
  - Hataları İşaretle: Sorunlu token'ları `UNKNOWN` olarak etiketle.
  - Konum Raporlama: Hata mesajları için doğru konum bilgisi sağla.

2.5 Performans Optimizasyonları

- **Verimli İşleme:**
  - Tek Geçiş: Kaynak kodunda tek bir yineleme.
  - Karakter-karakter: Geri izleme veya ileri bakma yok.
  - Hash Tablosu Aramaları: O(1) anahtar kelime ve operatör tanıma.
  - Minimum Dize İşlemleri: Mümkün olduğunca doğrudan karakter karşılaştırmaları.

3. Ayrıştırma Metodolojisi

3.1 Ayrıştırıcı Tasarım Felsefesi

- Yaklaşım: Basitleştirilmiş söz dizimi doğrulama ayrıştırıcısı.
- Tasarım Kararları:
  - Doğrulama Odaklılık: Tam bir AST (Abstract Syntax Tree) oluşturmaktan ziyade söz dizimi doğruluğunu kontrol et.
  - Hata Toplama: İlk hatada durmak yerine tüm hataları topla.
  - Hafif Uygulama: Gerçek zamanlı çalışma için minimum yük.
  - Eğitim Değeri: Temel ayrıştırma kavramlarını göster.

3.2 Ayrıştırma Algoritması

- **İki Aşamalı Yaklaşım:**
  - Token Filtreleme: Boşluk ve yeni satır token'larını kaldır.
  - Söz Dizimi Doğrulama: Yapısal doğruluğu kontrol et.

```python
def parse(self):
    """Parse the tokens (simplified)"""
    self.errors = []  # Önceki hataları temizle
    # Basit sözdizimi kontrolleri
    self._check_parentheses_balance()  # Parantez dengesi kontrolü
    self._check_string_completeness()  # Stringlerin doğru kapanıp kapanmadığı kontrolü
    return None  # Tam bir uygulamada burada AST dönerdi
```

3.3 Parantez/Köşeli Parantez Dengeleme

- Algoritma: Yığın tabanlı eşleştirme.

```
def _check_parentheses_balance(self):
    """Check if parentheses are balanced"""
    stack = []  # Açılan parantezleri saklamak için yığın
    pairs = {'(': ')', '[': ']', '{': '}'}  # Açılış-kapanış eşleşmeleri

    for token in self.tokens:
        if token.value in pairs:
            # Açılış parantezi ise yığına ekle
            stack.append((token.value, token.line, token.column))
        elif token.value in pairs.values():
            # Kapanış parantezi ise
            if not stack:
                # Yığında hiç açılış yoksa hata ekle
                self.errors.append(f"Unmatched closing '{token.value}' at line {token.line}, column {token.column}")
            else:
                opening, line, col = stack.pop()  # Son açılanı çıkar
                expected = pairs[opening]  # Beklenen kapanış karakteri
                if token.value != expected:
                    # Eşleşmiyorsa hata ekle
                    self.errors.append(f"Mismatched parentheses: expected '{expected}' but found '{token.value}' at lin

    for opening, line, col in stack:
        # Yığında kalan açılışlar varsa kapanmamış demektir
        self.errors.append(f"Unclosed '{opening}' at line {line}, column {col}")
```

- **Özellikler:**
  - İç İçe Yapı Desteği: Çoklu iç içe geçme seviyelerini işler.
  - Uyuşmazlık Tespiti: Yanlış kapanış parantez tiplerini belirler.
  - Konum Raporlama: Kesin hata konum bilgisi.
  - Kapsamlı Hata Raporlama: Tüm parantez sorunlarını raporlar.

3.4 Dize Tamamlama Doğrulaması

- Amaç: Dize değişmezlerinin düzgün şekilde kapatıldığını doğrula.

```
def _check_string_completeness(self):
    """Check if strings are properly closed"""
    for token in self.tokens:
        if token.type == TokenType.STRING:
            # Stringin başı ve sonu aynı karakter mi ve en az iki karakter mi?
            if len(token.value) < 2 or token.value[0] != token.value[-1]:
                self.errors.append(f"Unclosed string at line {token.line}, column {token.column}")
```

- **Doğrulama Kriterleri:**
  - Minimum Uzunluk: Dizeler en az açılış ve kapanış tırnaklarına sahip olmalıdır.
  - Tırnak Eşleştirme: İlk ve son karakterler eşleşmelidir.
  - Hata Raporlama: Sorunlu dizelerin konumunu raporlar.

3.5 Hata Kurtarma Stratejisi

- Felsefe: Kapsamlı geri bildirim için tüm hataları topla.
- Uygulama Yaklaşımı:
  - Hatada Devam Et: İlk hatada ayrıştırmayı durdurma.
  - Hata Birikimi: Daha sonra raporlama için hataları listede topla.
  - Bağlam Koruma: Hatalara rağmen ayrıştırma durumunu koru.

- Kullanıcı Dostu Mesajlar: Açık, eyleme geçirilebilir hata açıklamaları.

3.6 Gelecekteki Genişletilebilirlik

- **Ayrıştırıcı Genişletme Noktaları:**
  - Ek Söz Dizimi Kontrolleri: Yeni doğrulama kuralları eklemek kolay.
  - AST Oluşturma: Tam ayrıştırıcı uygulaması için hazır çerçeve.
  - Semantik Analiz: Tip kontrolü ve analizi için temel.
  - Hata Kurtarma: Daha gelişmiş kurtarma stratejileriyle geliştirilebilir.

4. Söz Dizimi Analiz Süreci

4.1 Analiz İş Akışı

- **Çok Aşamalı Süreç:**
  - Sözcüksel Analiz: Kod → Token'lar.
  - Söz Dizimi Ayrıştırma: Token'lar → Doğrulama Sonuçları.
  - İstatistik Üretimi: Analiz → Metrikler.
  - Görsel Güncelleme: Sonuçlar → GUI Sunumu.

```python
def _perform_analysis(self, code: str):
    """Leksik ve sözdizimsel analiz yapar"""
    self.lexer = SimpleLexicalAnalyzer(code)
    self.tokens = self.lexer.tokenize()
    self.parser = SimpleParser(self.tokens)
    self.ast = self.parser.parse()
```

4.2 Gerçek Zamanlı Analiz Entegrasyonu

- **Güncelleme Tetikleme:**

```python
def on_text_change(self, event=None):
    """Kullanıcı kodu değiştirdiğinde çağrılır (debounce ile)"""
    if not self.update_pending:
        self.update_pending = True
        self.root.after(300, self.update_highlighting)
```

- **Gecikme Stratejisi (Debouncing):**
  - Amaç: Hızlı yazma sırasında aşırı analizi önle.
  - Gecikme: Yanıt verebilirlik ve performans arasında 300ms denge.
  - Bayrak Sistemi: Çoklu bekleyen güncellemeleri önler.
  - Kullanıcı Deneyimi: Akıcı, engellemeyen etkileşim.

4.3 Analiz Koordinasyonu

- **Ana Analiz Kontrolcüsü:**

```python
def update_highlighting(self):
    """Gerçek zamanlı sözdizimi vurgulamasını günceller"""
    self.update_pending = False
    try:
        code = self.code_text.get("1.0", tk.END)
        self.status_manager.set_analyzing()
        self.root.update_idletasks()
        self._perform_analysis(code)
        self._update_all_views()
    except Exception as e:
        self.status_manager.set_error(str(e))
        messagebox.showerror("Analysis Error", f"An error occurred during analysis:\n{str(e)}")
```

- **Hata İşleme:** Analiz hatalarında zarif bozunma.

4.4 İstatistik Üretimi

- **Kapsamlı Metrik Toplama:**

```python
def update_statistics(self, lexer_stats: dict, parser_errors: List[str]):
    """İstatistik ekranını günceller"""
    stats_text = self._generate_statistics_text(lexer_stats, parser_errors)
    self.text_widget.config(state=tk.NORMAL)
    self.text_widget.delete("1.0", tk.END)
    self.text_widget.insert("1.0", stats_text)
    self.text_widget.config(state=tk.DISABLED)
```

- **Sağlanan Metrikler:**
  - Token Sayıları: Toplam token'lar ve tipe göre dağılım.
  - Satır İstatistikleri: İşlenen toplam satır sayısı.
  - Hata Sayıları: Sözcüksel ve söz dizimi hataları.
  - Yüzde Dağılımları: Token tiplerinin göreli sıklığı.

4.5 Görünüm Senkronizasyonu

- **Çoklu Görünüm Güncelleme Stratejisi:**

```python
def _update_all_views(self):
    """Tüm arayüzü analiz sonuçlarıyla günceller"""
    self.highlighted_view.update_highlighted_text(self.tokens)
    self.token_analysis_view.update_tokens(self.tokens)
    lexer_stats = self.lexer.get_statistics()
    parser_errors = self.parser.errors if self.parser else []
    self.statistics_view.update_statistics(lexer_stats, parser_errors)
    error_count = len(parser_errors)
    self.status_manager.set_complete(len(self.tokens), error_count)
```

- **Koordinasyon Özellikleri:**
  - Senkronize Güncellemeler: Tüm görünümler aynı analizden güncellenir.
  - Tutarlı Durum: Analiz sonuçları için tek bir doğru kaynak.
  - Durum İletişimi: Süreç boyunca kullanıcı geri bildirimi.

5. Vurgulama Şeması

5.1 Renk Tasarım Felsefesi

- Semantik Renk Eşleştirme: Kod yapısının anlaşılmasını pekiştirmek için seçilen renkler.
- **Renk Paleti:**

```
COLORS = {
    TokenType.KEYWORD: "#0066CC",      # Anahtar kelimeler için mavi
    TokenType.STRING: "#009900",       # Stringler için yeşil
    TokenType.NUMBER: "#FF6600",       # Sayılar için turuncu
    TokenType.COMMENT: "#808080",      # Yorumlar için gri
    TokenType.OPERATOR: "#CC0000",     # Operatörler için kırmızı
    TokenType.DELIMITER: "#663399",    # Ayraçlar için mor
    TokenType.IDENTIFIER: "#000000",   # Tanımlayıcılar için siyah
    TokenType.UNKNOWN: "#FF0000"       # Hatalı/unknown için parlak kırmızı
}
```

5.2 Renk Psikolojisi ve Konvansiyonlar

- **Tasarım Gerekçesi:**
  - Anahtar Kelimeler (Mavi):
    - Psikoloji: Otorite, istikrar, güven.
    - Konvansiyon: IDE'lerde dil yapıları için yaygın olarak kullanılır.
    - Fonksiyon: Kodun yapısal öğelerini vurgular.
  - Dizeler (Yeşil):
    - Psikoloji: Büyüme, veri, içerik.
    - Konvansiyon: Literal veriler için yaygın seçim.
    - Fonksiyon: Verileri kod mantığından ayırır.
  - Sayılar (Turuncu):
    - Psikoloji: Enerji, dikkat, değerler.
    - Konvansiyon: Literaller için sıcak renkler.
    - Fonksiyon: Sayısal değerleri öne çıkarır.
  - Yorumlar (Gri):
    - Psikoloji: İkincil, destekleyici bilgi.
    - Konvansiyon: Dokümantasyon için soluk renkler.
    - Fonksiyon: Yürütülemeyen metni görsel olarak önemsizleştirir.
  - Operatörler (Kırmızı):
    - Psikoloji: Eylem, işlem, dönüşüm.
    - Konvansiyon: İşlemler için dikkat çekici.
    - Fonksiyon: Eylemleri gerçekleştiren kodu vurgular.
  - Sınırlayıcılar (Mor):
    - Psikoloji: Yapı, organizasyon, gruplama.
    - Konvansiyon: Noktalama işaretleri için farklı renk.
    - Fonksiyon: Kod organizasyonunu ve kapsamını gösterir.

5.3 Erişilebilirlik Hususları

- **Tasarım Kriterleri:**

- Kontrast Oranları: Tüm renkler minimum kontrast gereksinimlerini karşılar.
- Renk Körlüğü: Renk şeması en yaygın renk körlüğü türleri için çalışır.
- Okunabilirlik: Renkler metin anlaşılmasını engellemez.
- Tutarlılık: Benzer semantik öğeler benzer renkleri kullanır.

5.4 Tipografi Geliştirme

- **Font Stili:**

```
def _setup_tags(self):
    """Vurgulama için tag'leri ayarla"""
    for token_type in TokenType:
        tag_name = token_type.value.lower()
        color = ColorScheme.get_color(token_type)
        self.text_widget.tag_configure(tag_name, foreground=color)

        # Anahtar kelimeler kalın, yorumlar italik
        if token_type == TokenType.KEYWORD:
            self.text_widget.tag_configure(tag_name, font=("Consolas", 12, "bold"))
        elif token_type == TokenType.COMMENT:
            self.text_widget.tag_configure(tag_name, font=("Consolas", 12, "italic"))
```

- **Tipografi Kararları:**
  - Anahtar Kelimeler: Vurgu için kalın ağırlık.
  - Yorumlar: Ayırt etmek için italik stil.
  - Monospace Yazı Tipi: Kod netliği için Consolas.
  - Tutarlı Boyutlandırma: Okunabilirlik için 12pt.

5.5 Vurgulama Uygulama Süreci

- **Token-by-Token Uygulama:**

```
def update_highlighted_text(self, tokens: List[Token]):
    """Vurgulu metni güncelle"""
    self.text_widget.config(state=tk.NORMAL)
    self.text_widget.delete("1.0", tk.END)
    for token in tokens:
        tag_name = token.type.value.lower()
        self.text_widget.insert(tk.END, token.value, tag_name)
    self.text_widget.config(state=tk.DISABLED)
```

- **Süreç Özellikleri:**
  - Kesin Uygulama: Her token ayrı ayrı biçimlendirilir.
  - Etiket Tabanlı Sistem: Esnek stil için Tkinter etiketleri.
  - Dinamik Güncellemeler: Gerçek zamanlı vurgulama değişiklikleri.
  - Koruma: Salt okunur ekran, kullanıcı değişikliklerini önler.

5.6 Performans Optimizasyonu

- **Verimli Vurgulama:**
  - Tek Geçiş: Görüntüleme için token'larda tek bir yineleme.
  - Etiket Yeniden Kullanımı: Etiketler bir kez oluşturulur, birçok kez uygulanır.
  - Toplu Güncellemeler: Tüm değişiklikler tek bir işlemde uygulanır.

- Minimum Yeniden Çizimler: Yalnızca içerik değiştiğinde güncelle.

6. GUI Uygulaması

6.1 Çerçeve Seçimi ve Gerekçesi

- Seçilen Çerçeve: Tkinter.
- **Seçim Kriterleri:**
  - Dahili Kullanılabilirlik: Harici bağımlılık yok.
  - Çapraz Platform Desteği: Windows, macOS, Linux uyumluluğu.
  - Eğitim Uygunluğu: Python GUI eğitimi için standart.
  - Yeterli İşlevsellik: Proje gereksinimleri için yeterli widget'lar.
  - Dokümantasyon Kalitesi: Kapsamlı örneklerle iyi belgelenmiş.
- **Alternatif Hususlar:**
  - PyQt/PySide: Daha fazla özellik, ancak harici bağımlılık.
  - Kivy: Modern dokunmatik arayüz, ancak öğrenme eğrisi.
  - Web tabanlı: HTML/CSS esnekliği, ancak dağıtım karmaşıklığı.

6.2 Uygulama Mimarisi

- **Ana Pencere Yapısı:**

```
def setup_gui(self):
    """Arayüz bileşenlerini kurar"""
    main_frame = ttk.Frame(self.root)
    main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
    notebook = ttk.Notebook(main_frame)
    notebook.pack(fill=tk.BOTH, expand=True)
    self._setup_editor_tab(notebook)
    self._setup_analysis_tab(notebook)
    self._setup_statistics_tab(notebook)
    self.status_var = tk.StringVar()
    self.status_manager = StatusManager(self.status_var)
    self.status_manager.set_status("Ready")
    status_bar = ttk.Label(main_frame, textvariable=self.status_var, relief=tk.SUNKEN)
    status_bar.pack(side=tk.
                    (parameter) notebook: Any
```

- **Mimari Kararlar:**
  - Defter Arayüzü: Sekmeli bölümlerle düzenli iş akışı.
  - Hiyerarşik Düzen: İlgili işlevselliğin mantıksal gruplandırılması.
  - Durum İletişimi: Kullanıcı geri bildirimi için her zaman görünür durum çubuğu.
  - Duyarlı Tasarım: Pencere boyutlandırma için `pack/fill`'in doğru kullanımı.

6.3 Kod Düzenleyici Uygulaması

- **Bölünmüş Bölme Tasarımı:**

```python
def _setup_editor_tab(self, notebook):
    """Kod editörü sekmesini kurar"""
    editor_frame = ttk.Frame(notebook)
    notebook.add(editor_frame, text="Code Editor")
    paned = ttk.PanedWindow(editor_frame, orient=tk.HORIZONTAL)
    paned.pack(fill=tk.BOTH, expand=True)
    left_frame = ttk.LabelFrame(paned, text="Python Code Input", padding=5)
    paned.add(left_frame, weight=1)
    self.code_text = scrolledtext.ScrolledText(
        left_frame,
        wrap=tk.NONE,
        font=("Consolas", 12),
        undo=True,
        width=50,
        height=30
    )
    self.code_text.pack(fill=tk.BOTH, expand=True)
    right_frame = ttk.LabelFrame(paned, text="Syntax Highlighted Output", padding=5)
    paned.add(right_frame, weight=1)
    self.highlighted_view = HighlightedTextView(right_frame)
    self._load_sample_code()
```

- **Tasarım Özellikleri:**
  - Yan Yana Karşılaştırma: Giriş ve çıkış aynı anda görünür.
  - Yeniden Boyutlandırılabilir Paneller: Kullanıcı göreli boyutları ayarlayabilir.
  - Açık Etiketleme: Açıklayıcı başlıklara sahip ayrı bölümler.
  - Eşit Ağırlık Dağılımı: Dengeli başlangıç düzeni.

6.4 Metin Widget Yapılandırması

- **Giriş Metni Widget'ı:**

```python
self.code_text = scrolledtext.ScrolledText(
    left_frame,
    wrap=tk.NONE,
    font=("Consolas", 12),
    undo=True,
    width=50,
    height=30
)
```

- **Çıkış Metni Widget'ı:**

```python
self.text_widget = scrolledtext.ScrolledText(
    parent_frame,
    wrap=tk.NONE,
    font=("Consolas", 12),
    state=tk.DISABLED,
    width=50,
    height=30
)
```

- **Yapılandırma Gerekçesi:**
  - Consolas Yazı Tipi: Doğru kod hizalaması ve okunabilirlik.
  - Kaydırma Yok: Kod biçimlendirmesini korur.
  - Kaydırma Desteği: Büyük kod dosyalarını işler.

- Geri Alma Desteği: Standart düzenleyici işlevselliği.
- Salt Okunur Çıkış: Yanlışlıkla yapılan değişiklikleri önler.

6.5 Token Analiz Tablosu

- **TreeView Uygulaması:**

```
def setup_treeview(self, parent_frame):
    """Token analiz ağacını oluşturur"""
    columns = ("Position", "Line", "Column", "Type", "Value")
    self.tree = ttk.Treeview(parent_frame, columns=columns, show="headings", height=20)
    for col in columns:
        self.tree.heading(col, text=col)
        self.tree.column(col, width=100)
    scrollbar = ttk.Scrollbar(parent_frame, orient=tk.VERTICAL, command=self.tree.yview)
    self.tree.configure(yscrollcommand=scrollbar.set)
    self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
```

- **Tablo Özellikleri:**
  - Yapılandırılmış Görüntü: Tüm token nitelikleri için sütunlar.
  - Sıralanabilir Başlıklar: Herhangi bir sütuna göre sıralamak için tıkla.
  - Kaydırılabilir İçerik: Büyük token listelerini işler.
  - Detaylı Bilgi: Tam token meta verileri görünür.

6.6 İstatistik Görüntüleme

- **Zengin Metin Uygulaması:**

```
self.text_widget = scrolledtext.ScrolledText(
    parent_frame,
    height=20,
    width=80,
    state=tk.DISABLED
)
```

- **İçerik Üretimi:**

```
def _generate_statistics_text(self, lexer_stats: dict, parser_errors: List[str]) -> str:
    """İstatistik metnini oluşturur"""
    if not lexer_stats:
        return "No analysis data available."
    stats_text = "=== LEXICAL ANALYSIS STATISTICS ===\n\n"
    stats_text += f"Total Tokens: {lexer_stats.get('total_tokens', 0)}\n"
    stats_text += f"Total Lines: {lexer_stats.get('total_lines', 0)}\n\n"
    stats_text += "Token Type Distribution:\n"
    stats_text += "-" * 30 + "\n"
    token_counts = lexer_stats.get('token_counts', {})
    total_tokens = lexer_stats.get('total_tokens', 1)
    for token_type, count in sorted(token_counts.items(), key=lambda x: x[1], reverse=True):
        percentage = (count / total_tokens) * 100
        stats_text += f"{token_type.value:<15}: {count:>6} ({percentage:>5.1f}%)\n"
    if parser_errors:
        stats_text += "\n=== PARSE ERRORS ===\n\n"
        for error in parser_errors:
            stats_text += f"• {error}\n"
    else:
        stats_text += "\nNo parse errors detected.\n"
    if lexer_stats.get('error_count', 0) > 0:
        stats_text += f"\nTotal Lexical Errors: {lexer_stats.get('error_count', 0)}\n"
    else:
        stats_text += "\nNo lexical errors detected.\n"
    if lexer_stats.get('warning_count', 0) > 0:
        stats_text += f"Total Warnings: {lexer_stats.get('warning_count', 0)}\n"
    else:
        stats_text += "No warnings detected.\n"
    return stats_text
```

- **İstatistik Özellikleri:**
  - Kapsamlı Metrikler: Tam analiz özeti.
  - Biçimlendirilmiş Çıktı: Profesyonel rapor görünümü.
  - Yüzde Hesaplamaları: Göreli sıklık bilgisi.
  - Hata Entegrasyonu: Ayrıştırma hataları rapora dahil edilir.

6.7 Olay Sistemi Uygulaması

- **Gerçek Zamanlı Güncelleme Bağlama:**

```
def _setup_statistics_tab(self, notebook):
    """İstatistik sekmesini kurar"""
    stats_frame = ttk.Frame(notebook)
    notebook.add(stats_frame, text="Statistics")
    self.statistics_view = StatisticsView(stats_frame)
```

- **Olay İşleme:**

```
def on_text_change(self, event=None):
    """Kullanıcı kodu değiştirdiğinde çağrılır (debounce ile)"""
    if not self.update_pending:
        self.update_pending = True
        self.root.after(300, self.update_highlighting)
```

- **Olay Sistemi Özellikleri:**
  - Çoklu Tetikleyiciler: Klavye ve fare olayları.
  - Gecikmeli İşleme (Debounced Processing): Aşırı güncellemeleri önler.
  - Bayrak Tabanlı Kontrol: Çoklu bekleyen güncellemeleri önler.
  - Zamanlayıcı Entegrasyonu: Tkinter'ın dahili zamanlayıcı sistemini kullanır.

6.8 Durum Yönetim Sistemi

- **Durum İletişimi:**

```
class StatusManager:
    """Durum çubuğu güncellemelerini yönetir"""

    def __init__(self, status_var: tk.StringVar):
        self.status_var = status_var

    def set_status(self, message: str):
        """Durum mesajını ayarla"""
        self.status_var.set(message)

    def set_analyzing(self):
        """Analiz ediliyor mesajı"""
        self.set_status("Analyzing...")

    def set_complete(self, token_count: int, error_count: int = 0):
        """Analiz tamamlandı mesajı"""
        if error_count > 0:
            self.set_status(f"Analysis complete - {token_count} tokens, {error_count} parse errors")
        else:
            self.set_status(f"Analysis complete - {token_count} tokens")

    def set_error(self, error_msg: str):
        """Hata mesajı"""
        self.set_status(f"Error: {error_msg}")
```

- **Durum Özellikleri:**
  - Merkezi Yönetim: Durum güncellemeleri için tek nokta.
  - Bağlamsal Mesajlar: Farklı durumlar için farklı mesajlar.
  - Kullanıcı Geri Bildirimi: Her zaman görünür ilerleme göstergesi.
  - Hata İletişimi: Açık hata raporlama.

6.9 Performans Optimizasyonları

- **GUI Performans Stratejileri:**
  - Sınırlı Token Görüntüleme: Analiz tablosunda maksimum 500 token.
  - Gecikmeli Güncellemeler (Debounced Updates): Aşırı işlemeyi önleyen 300ms gecikme.
  - Durum Yönetimi: Salt okunur içerik için devre dışı bırakılmış widget'lar.
  - Toplu İşlemler: Birden çok değişiklik birlikte uygulanır.
- **Bellek Yönetimi:**
  - Token Limitleme: Büyük dosyalarla bellek sorunlarını önler.
  - Widget Yeniden Kullanımı: Etiketler ve widget'lar yeniden oluşturulmak yerine yeniden kullanılır.
  - Verimli Güncellemeler: Yalnızca değişen içerik yenilenir.

6.10 Kullanıcı Deneyimi Tasarımı

- **Kullanılabilirlik Özellikleri:**
  - Anında Geri Bildirim: Kullanıcı yazdıkça gerçek zamanlı vurgulama.
  - Çoklu Görünümler: Aynı analiz üzerinde farklı bakış açıları.
  - Açık Gezinme: Kolay geçiş için sekmeli arayüz.
  - Kapsamlı Bilgi: Detaylı analiz sonuçları mevcut.
  - Hata Görünürlüğü: Sorunlar açıkça vurgulanır ve raporlanır.
- **Erişilebilirlik Hususları:**
  - Klavye Navigasyonu: Tüm işlevsellik klavye ile erişilebilir.
  - Açık Etiketler: Tüm arayüz öğeleri için açıklayıcı metin.
  - Tutarlı Düzen: Tahmin edilebilir arayüz organizasyonu.
  - Durum İletişimi: Uygulamanın ne yaptığını her zaman açıkça belirtir.

Sonuç

Bu Python Söz Dizimi Vurgulayıcı, dil işleme eğitimine kapsamlı bir yaklaşım sergilemektedir; teorik kavramları pratik uygulamayla birleştirir. Modüler tasarım, kapsamlı hata işleme ve sezgisel kullanıcı arayüzü, sözcüksel analiz, ayrıştırma ve söz dizimi vurgulama kavramlarını anlamak için etkili bir araç oluşturur. Burada belgelenen teknik kararlar, eğitim değeri, uygulama karmaşıklığı ve kullanıcı deneyimi dikkatli bir şekilde değerlendirilerek, akademik titizliği pratik kullanışlılıkla başarıyla dengeleyen bir sistemle sonuçlanmıştır.