

# PySyntaxHighlight Projesi Dökümantasyonu

## 1. Proje Hakkında Genel Bilgi

PySyntaxHighlight, Python ve Tkinter kullanılarak geliştirilen bir gerçek zamanlı sözdizimi vurgulayıcı uygulamadır. Bu proje, kullanıcıların Python kodlarını yazarken belirteç türlerini anında farklı renklerle görebileceği bir platform sunar. Örneğin, bir `def` anahtar kelimesi yazıldığında bu kelime hemen mavi renkte görünür. Uygulama, açık ve koyu tema seçeneklerine sahiptir ve metin düzenleme için standart Text bileşeni kullanılır. Vurgulama mantığı tamamen projeye özel olarak tasarlanmıştır; herhangi bir hazır kütüphane kullanılmaz.

## 2. Dil ve Dilbilgisi Seçimi

Proje için Python programlama dili tercih edilmiştir. Sebepleri şunlardır: - Sade ve okunabilir sözdizimi - Tkinter ile GUI geliştirme kolaylığı - Nesne yönelimli programlama desteği

Dilbilgisi olarak Python'un temel yapıları desteklenmektedir. Aşağıdaki gibi yapılar vurgulanabilir:

```
def foo(x):
    if x > 0:
        return x
```

Desteklenen yapılar: - Anahtar kelimeler: `def`, `if`, `return`, `class`, `for`, `while` - Sayılar: 42, 3.14 - String'ler: "merhaba", 'dünya' - F-string: `f"Sonuç: {x}"` - Operatörler: +, -, \*, /, == - Yorumlar: # bu bir yorum

## 3. Sözdizimi Analizi Süreci

### 3.1 Leksikal Analiz

Leksikal analiz, kullanıcı tarafından yazılan kodu satır satır ve karakter karakter inceleyerek anlamlı küçük parçalara (token) ayırma işlemidir. Bu işlem, Lexer sınıfı tarafından gerçekleştirilir.

#### Kullanılan Yapılar:

- `advance()`: Mevcut karakteri bir sonraki karaktere taşır.
- `set_text(text)`: Tokenlaştırılacak metni yükler.
- `tokenize()`: Metni tarar ve token listesini oluşturur.

#### Desteklenen Token Türleri:

Token Türü	Açıklama
KEYWORD	<code>def</code> , <code>return</code> , <code>if</code> gibi Python sözcükleri
IDENTIFIER	Fonksiyon, değişken adları

Token Türü	Açıklama
NUMBER	123, 3.14 gibi sayılar
STRING	"merhaba" gibi sabit metinler
FSTRING	f"değer: {x}" türünde diziler
FSTRING_EXPR	F-string içindeki gömülü ifadeler
OPERATOR	+, =, ==, : gibi operatörler
COMMENT	# bu bir yorum
ERROR	Geçersiz karakterler veya yapılar

#### Örnek Algoritma Akışı

```
text = "def topla(x, y): return x + y"
lexer = Lexer()
lexer.set_text(text)
tokens = lexer.tokenize()
```

Cıktı (token listesi):

```
[KEYWORD:def, IDENTIFIER:topla, OPERATOR:(, IDENTIFIER:x, ...,
KEYWORD:return, IDENTIFIER:x, OPERATOR:+, IDENTIFIER:y]
```

#### Özel Durumlar:

- Üçlü tırnak (""" veya ''') ile yazılmış çok satırlı yorumlar/strings
- F-string içinde {} içindeki ifadeler, FSTRING\_EXPR olarak ayrılır
- Arka arkaya gelen kapalı parantezler gibi hatalar ERROR ile işaretlenir

## 3.2 Söz dizimi Analizi

Leksikal analiz sonucu elde edilen token'lar, Parser sınıfı tarafından yapılandırılır. Buradaki amaç, token'ları Python dil kurallarına göre anlamlı yapılar (fonksiyonlar, sınıflar, ifadeler) haline getirmektir.

#### Ana Yöntemler:

Metot	Görev
parse()	Tüm kodu işler, deyimleri toplar
parse_function_def()	def ile başlayan fonksiyonları işler
parse_if_stmt()	if, elif, else bloklarını çözümler
parse_expression()	Ifadeleri analiz eder (aritmetik, karşılaştırma vb.)

#### Örnek Gramer Kuralları (CFG temelli):

```
FunctionDef → 'def' IDENTIFIER '(' [ParamList] ')' ':' Suite
IfStmt → 'if' Expression ':' Suite [elif/else...]
Assignment → IDENTIFIER '=' Expression
Expression → Term { ('+' | '-') Term }
```

### ☞ Örnek Kullanım

```
lexer = Lexer()
lexer.set_text("def foo(x): return x + 1")
tokens = lexer.tokenize()

parser = Parser(tokens)
statements = parser.parse()
```

Çıktı:

```
[ ('FUNCTION_DEF', 'foo', ['x'], [
    ('RETURN_STMT', ('ARITH', ('IDENTIFIER', 'x'), '+', ('LITERAL', '1'))),
    [], False)
]
```

### ⌚ Parser'ın Özellikleri:

- Her token tipi için uygun `parse_*` metodu vardır.
- `try/except/finally`, `match/case`, `lambda` gibi yapılar desteklenir.
- `max_iterations` gibi önlem mekanizmaları içerir (sonsuz döngü önlemi).
- Parser çıktısı GUI'de vurgulama yaparken daha derin analiz sağlar.

## 4. Vurgulama Sistemi

Kod vurgulama işlemi `highlight_syntax()` fonksiyonu ile gerçek zamanlı yapılır. Kullanıcı yazı yazdığında: - Lexer çağrıları ve token'lar üretilir. - Parser çağrıları ve yapı analizi yapılır. - Etiketler, Text widget'ındaki pozisyonlara uygulanır.

### Temaya Göre Renkler (Koyu Tema)

Token Türü	Renk
KEYWORD	Açık mavi
STRING	Somon
NUMBER	Açık yeşil
COMMENT	Yumuşak yeşil
IDENTIFIER	Açık gri
FSTRING_EXPR	Turkuaz
FUNCTION_DEF	Sarımsı
ERROR	Kırmızı

Ayrıca fonksiyon parametreleri, çağrılar, sınıf adları ve lambda ifadeleri de özel olarak renklendirilir.

## 5. Grafik Arayüz (GUI) Tasarımı

Arayüz, Tkinter kullanılarak tasarlanmıştır.

#### Ana bileşenler:

- **Metin Alanı:** Kodun yazıldığı yer. Gerçek zamanlı vurgulama uygulanır.
- **Satır Numaraları:** Text widget'ı ile senkronize çalışan sol alan.
- **Tema Düğmesi:** Açık/koyu tema geçişini sağlar.
- **Dosya Menüleri:** Kod dosyası açma, kaydetme, farklı kaydetme.

Uygulama, kullanıcının metin alanı ile etkileşimde bulunduğu her an analiz ve vurgulama işlemlerini yeniden çalıştırır.

## 6. Kod Yapısı

Proje üç ana bölümden oluşur:

- **Lexer sınıfı:** Karakter girdilerini token'lara çevirir.
- **Parser sınıfı:** Token'ları yapı olarak çözümler.
- **SyntaxHighlighterGUI sınıfı:** GUI ve vurgulama işlemleri

Giriş noktası:

```
if __name__ == "__main__":
    root = tk.Tk()
    app = SyntaxHighlighterGUI(root)
    root.mainloop()
```

## 7. Sonuç

PySyntaxHighlight, gerçek zamanlı sözdizimi vurgulama ve analiz yetenekleri ile Python geliştiricilerine güçlü bir araç sunar. Proje sıfırdan yazılmış Lexer ve Parser bileşenleriyle hazır kütüphanelerden bağımsız olarak çalışır.

Ayrıca açık ve koyu tema desteği ile kullanıcı dostu bir deneyim sağlar. Gelecekteki planlar arasında: - Hata mesajları gösterimi - Kod tamamlama - Genişletilmiş gramer desteği gibi özellikler bulunmaktadır.