BURSA TEKNİK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROGRAMLAMA DİLLERİ

SYNTAX HIGHLIGHTER

22360859002

SUDENUR ELMAS

Real-Time Grammar-Based Syntax Highlighter with GUI

1. Language and Grammar Choice

Seçilen Dil: Python

1.1 Neden Python?

Bu projede Python dilinin tercih edilmesinin başlıca nedenleri:

- **Popülerlik ve Sadelik:** Python, temiz ve okunabilir sözdizimi sayesinde eğitim ve prototipleme projeleri için ideal bir dildir.
- **Modülerlik:** Python'un yapılandırılmış ve nesne yönelimli özellikleri, Lexer ve Parser gibi modüllerin net bir şekilde ayrılmasını sağlar.
- **Geniş Topluluk ve Belgelendirme:** Python dilinin resmi belgeleri ve topluluk desteği, gramer kurallarının açık bir şekilde tanımlanmasını kolaylaştırır.

1.2 Dilbilgisi Yapısı

Proje, Python'un temel sözdizimini tanıyan ve *Context-Free Grammar (CFG)* yapısına dayanan bir parser ile gerçekleştirilmiştir. Gramer, recursive descent yöntemiyle yorumlanmış ve parse_ fonksiyonları yardımıyla ayrıştırma yapılmıştır.

Grammar: Python dilinin temel sözdizimi kuralları temel alınmıştır. Özellikle:

- if, else, while, for, def, return, class gibi yapılar,
- Değişken atamaları ve aritmetik işlemler,
- Fonksiyon tanımları ve çağrıları,
- Yorum satırları,
- F-string ve string tanımları.

2. Syntax Analysis Process

Amaç: Kullanıcının yazdığı Python kodunu anlık olarak tokenize etmek ve bu token'lardan AST üretmektir.

Aşamalar:

1. Kaynak Kod: Kullanıcının editöre girdiği Python metni.
2. Lexer : Karakterleri token'lara ayırır.

3. Parser: Tokenları CFG kurallarına göre analiz ederek AST (Abstract Syntax Tree) oluşturur.
4. Renklendirme: Token türüne göre renkli vurgulama yapılır.
5. Görselleştirme: Parse sonucu ve AST, GUI üzerinde anlık olarak kullanıcıya sunulur.

Bu işlem, yazılan kodda herhangi bir değişiklik olduğunda tetiklenerek anlık (real-time) olarak gerçekleştirilir.

Proje Python'un temel sözdizimini tanıyan ve *Context-Free Grammar (CFG)* yapısına dayanan bir parser ile gerçekleştirilmiştir. Desteklenen gramer kuralları:

```
<program>     ::= <statement>*
<statement>   ::= <assign_stmt>
                | <if_stmt>
                | <while_stmt>
                | <func_def>
                | <return_stmt>
                | <print_stmt>
                | <expr_stmt>
<assign_stmt>  ::= IDENTIFIER "=" <expression>
<if_stmt>     ::= "if" <expression> ":" <block> ["elif" <expression> ":" <block>]* ["else" ":" <block>]
<while_stmt>   ::= "while" <expression> ":" <block>
<func_def>    ::= "def" IDENTIFIER "(" [<param_list>] ")" ":" <block>
<param_list>   ::= IDENTIFIER ("," IDENTIFIER)*
<return_stmt>  ::= "return" [<expression>]
<print_stmt>   ::= "print" "(" [<expression> ("," <expression>)*] ")"
<expr_stmt>    ::= <expression>
<block>       ::= INDENT <statement>+ DEDENT | <statement>
<expression>   ::= <comparison> (("and" | "or") <comparison>)*
<comparison>   ::= <addition> (("==" | "!=" | "<" | ">" | "<=" | ">=") <addition>)*
<addition>    ::= <term> (("+" | "-") <term>)*
<term>        ::= <factor> (("*" | "/") <factor>)*
```

<factor> ::= NUMBER

| STRING

| IDENTIFIER

| IDENTIFIER "(" [<arg_list>] ")"

| "(" <expression> ")"

| "not" <factor>

| "[" [<expression_list>] "]"

| "{" [<keyvalue_list>]}"

<arg_list> ::= <expression> ("," <expression>)*

<expression_list> ::= <expression> ("," <expression>)*

<keyvalue_list> ::= STRING ":" <expression> ("," STRING ":" <expression>)*

3. Lexical Analysis Details

3.1 Kullanılan Yöntem: *State Diagram & Program Implementation*

Uygulama:

- Lexer sınıfı, Python'daki sabit anahtar kelimeleri (keywords), operatörleri (operators), ayraçları (delimiters) ve yerleşik fonksiyonları (builtins) tanımlar.
- Her karakterin tipi (boşluk, sayı, harf, vs.) ayrı fonksiyonlarla tespit edilir.
- Girdi metni baştan sona işlenerek:
    - Yorumlar (# ile başlayan),
    - Sayılar (tam ve ondalıklı),
    - Stringler (tek, çift, üçlü tırnaklar, f-string),
    - Operatörler ve ayraçlar,
    - Tanımlayıcılar ve anahtar kelimeler gibi yapılar token’lara dönüştürülür.

Token Sınıfları: "KEYWORD", "BUILTIN", "IDENTIFIER", "NUMBER", "STRING", "FSTRING", "COMMENT", "OPERATOR", "DELIMITER".

4. Parsing Methodology

4.1 Kullanılan Yöntem: Recursive Descent Parser

Her yapı için özel parse_ fonksiyonları yazılmıştır. Örneğin:

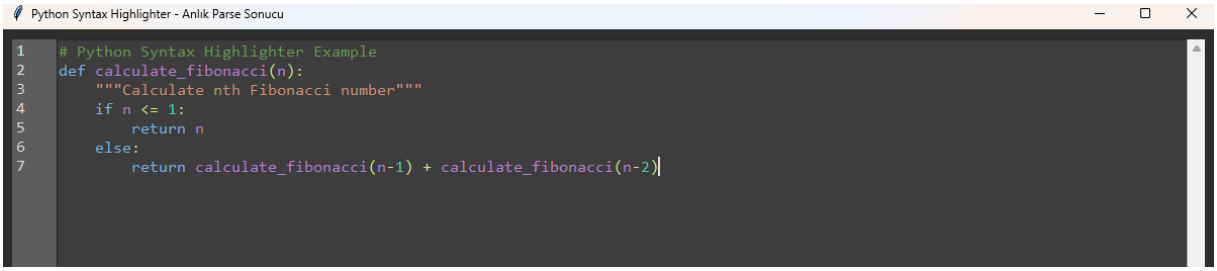
- parse_if_stmt()
- parse_func_def()
- parse_assignment()

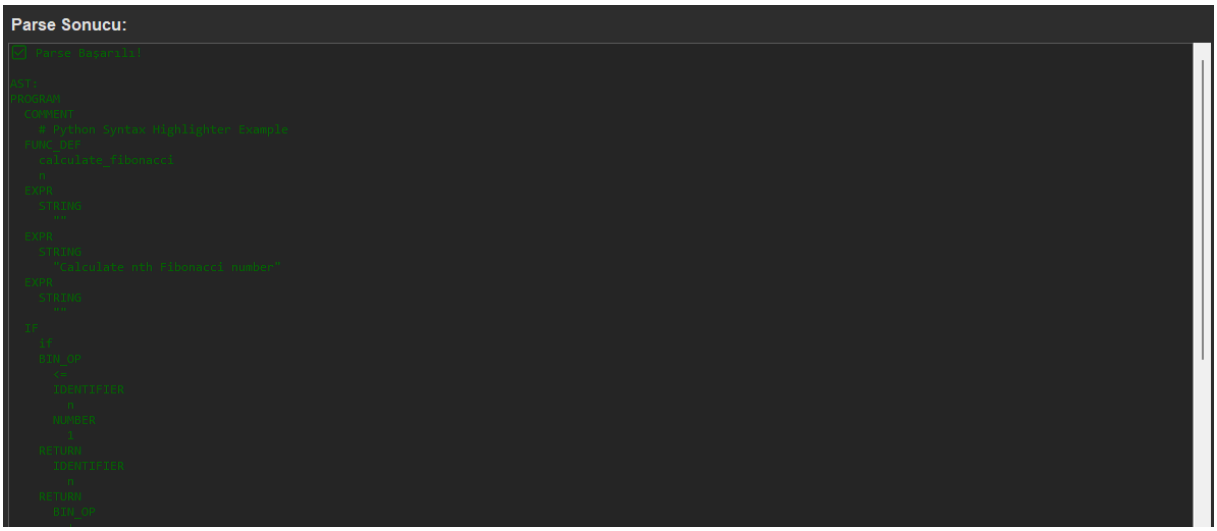
4.2 AST (Abstract Syntax Tree)

- Parser çıktısı, mantıksal kod yapısını temsil eden AST'tir.

Örnek bir AST düğümü:

```
('FUNC_DEF', 'calculate_fibonacci', ['n'], [...])
```





4.3 Hata Yönetimi

- expect() fonksiyonu ile beklenen tokenlar kontrol edilir.
- Hatalı durumda satır/pozisyon bilgisiyle kullanıcı uyarılır.
- GUI üzerinden parse hatası açıkça belirtilir.

5. Highlighting Scheme

5.1 Token Bazlı Vurgulama

Renklendirme, SyntaxHighlighter sınıfı aracılığıyla yapılır. Her token türü için Tkinter tag'ları tanımlanmıştır. Örnek renkler:

Token Türü	Renk (Dark Mode)
KEYWORD	Mavi (#7ab6e7)
BUILTIN	Mor (#7c75d6)
IDENTIFIER	Açık Mor (#bc80d8)
NUMBER	Turkuaz (#4ec9b0)
STRING	Somon (#ce9178)
FSTRING	Açık Mavi (#caf7f4)
COMMENT	Yeşil (#6a9955)
OPERATOR	Sarı (#d5e98f)
DELIMITER	Sarı (#d5e98f)

5.2 Gerçek Zamanlı (Real-Time) Renklendirme

- Kullanıcı kodda değişiklik yaptığında (klavye tuşu bırakıldığında, yapıştırma veya kesme işlemlerinde), on_modified() fonksiyonu tetiklenir.
- Bu fonksiyon:
    - Lexer ile metni tokenize eder,
    - Parser ile AST üretir,
    - Token pozisyonlarına göre renklendirme (tag_add) uygular,
    - Parse sonucu başarılıysa AST'yi, değilse hata mesajını gösterir.
- Ayrıca <<Change>> event'i manuel olarak gönderilerek satır numaralarının da güncellenmesi sağlanır.

6. GUI Implementation

6.1 Kullanılan Teknoloji ve Bileşenler

- **Tkinter:** Python'un yerleşik GUI kütüphanesi. Tüm arayüz bu kütüphane ile oluşturulmuştur.
- **Kod Editörü:** CustomText sınıfı ile özelleştirilmiş Text widget'ı kullanılmıştır. Yazı alanı üzerinde renklendirme ve değişiklik takibi yapılır.
- **Satır Numaraları:** LineNumbers sınıfı ile sol kenarda dinamik olarak gösterilir. Kullanıcının scroll hareketine senkronize çalışır.
- **Parse Sonucu Paneli:** Ekranın alt kısmında yer alır. Kod geçerliyse AST gösterilir, hata varsa hata mesajı ve token pozisyonu yazılır.
- **Anlık Güncelleme:** <KeyRelease>, <<Paste>>, <<Cut>> event'leriyle tetiklenir. Böylece kod yazımı sırasında anında hem renklendirme yapılır hem de AST güncellenir.
- **Dark Mode:** Karanlık tema desteklenmektedir. Renkler yukarıdaki tabloya göre ayarlanmıştır.

6.2 Örnek Kod Girdisi

- Arayüz açıldığında kullanıcıya örnek bir kod otomatik olarak sunulur:

```
def calculate_fibonacci(n):
    if n <= 1:
        return n
    else:
        return calculate_fibonacci(n-1) + calculate_fibonacci(n-2)
```

7. SONUÇ

Bu projede, Python programlama dilinin temel sözdizimini gerçek zamanlı olarak analiz eden ve renklendiren bir syntax highlighter geliştirilmiştir. Proje, Lexer (Sözcük Çözümleyici), Parser (Sözdizim Analizcisi), Real-time Syntax Highlighter ve GUI (Grafik Kullanıcı Arayüzü) bileşenlerinden oluşmaktadır.

✓ Lexer (Tokenization):

- Python kodunu token'lara ayırarak anahtar kelimeler, operatörler, yorumlar, sayılar ve string'leri doğru şekilde tanımladı.
- Farklı token türleri için renklendirme şeması uygulandı.

✓ **Parser (Top-Down):**

- Recursive Descent Parsing yöntemiyle Python'un temel gramer yapısı çözümlendi.
- Abstract Syntax Tree (AST) oluşturularak kodun yapısal analizi sağlandı.
- Hata yönetimi sayesinde geçersiz sözdizimleri tespit edilip kullanıcıya bildirildi.

✓ **Syntax Highlighting :**

- 9 farklı token türü için real-time renklendirme yapılarak kullanıcıya estetik bir deneyim sağlandı.

✓ **GUI:**

- Tkinter kullanılarak gerçek zamanlı renklendirme ve AST görselleştirme sağlandı.
- Karanlık mod desteğiyle modern bir IDE deneyimi sunuldu.
- Dinamik satır numaralandırma ve anlık güncelleme özellikleri eklendi.