

PROJE TANIMI : Gerçek zamanlı gramer tabanlı sözdizimi vurgulayıcı

PROJEYİ YAPAN KİŞİNİN ADI-SOYADI: Tuğba Nur Ayık

ÖĞRENCİ NUMARASI: 22360859035

## AMAÇ

---

Bu proje, kullanıcıların kendi tanımlı basit bir programlama dili ile yazdıkları kodları, gerçek zamanlı olarak hem renklendirme hem de sözdizimsel olarak doğrulama işlevi ile kontrol etmelerini sağlar. Hatalar anında GUI üzerinde belirtilir.

## PROJE KAPSAMINDA KULLANILAN DİL

---

Ben bu proje de daha hakim olduğum bir dil olduğu için Python dilini kullanmayı tercih ettim.

## Modüller ve Görevleri

---

### **lexer\_module.py**

Kodları analiz ederek anahtar kelimeler, operatörler, sayı, string gibi yapıları tanımlar ve token listesi oluşturur. Yani girilen kaynak kodu **token** denilen daha küçük anlamlı parçalara ayrılır. Bu işlem “lexical analysis” olarak bilinir.

### **Başlıca token türleri:**

- keywords = {'var', 'print', 'if', 'else', 'while'}
- IDENTIFIER: Kullanıcı tanımlı değişken adları
- NUMBER: Sayılar
- STRING: "tırnak" içindeki metinler
- OPERATOR: +, -, ==, =, >= vb.
- DELIMITER: ;, (, ), {, }

### **Char\_type (char)**

Her karakterin tipini belirler:

- Harf: 'letter'
- Rakam: 'digit'
- Boşluk: 'whitespace'
- Diğer durumlara göre özel karakterler (=, ;, ( vs.)

## Hatalar:

- Tanımsız karakter
- Bitmeyen string literal (örneğin "selam)
- Geçersiz operatör (örneğin !)

## lexer (code)

Giriş olarak verilen kodu **token listesine** dönüştürür.

Örnek çıktı:

```
[('KEYWORD', 'var'), ('IDENTIFIER', 'a'), ('OPERATOR', '='), ('NUMBER', '3')]
```

## main\_parser.py

Token listesini alır, dilin BNF yapısına uygunluğunu kontrol eder. Yani parser, lexer'dan gelen token dizisini kurallara uygun olarak inceler ve anlamlı yapıların doğru olup olmadığını kontrol eder.

Parser Sınıfı

Tokenizer'dan gelen token listesini alır ve üzerinde pozisyonla (self.pos) gezinir.

## Önemli Fonksiyonlar:

### eat(expected\_type, expected\_value=None)

Beklenen token türünü (ve varsa değerini) kontrol eder.

- Uygun değilse SyntaxError fırlatır.
- Özel olarak geçerli operatörleri sınırlar:

```
allowed_ops = ['+', '-', '*', '/', '==', '!=', '<', '>', '<=', '>=', '=', '++', '--']
```

parse()

Programın tamamının kurallara uygunluğunu kontrol eder.

## Dilin Desteklediği Yapılar (BNF tarzında):

### 1. var\_declaration – Değişken tanımı

```
var_declaration ::= "var" IDENTIFIER "=" expression ";"
```

### 2. print\_statement

```
print_statement ::= "print" "(" expression ")" ";"
```

### 3. if\_statement

if\_statement ::= "if" "(" expression ")" "{" statement\_list "}" ";"

### 4. while\_statement

while\_statement ::= "while" "(" expression ")" "{" statement\_list "}"

### 5.else\_statement

else\_statement ::= "else" "{" statement\_list "}"

## **İfade Analizi:**

Sıralı olarak parçalara ayrılır:

- expression → comparison (örneğin: a >= 7)
- comparison → term OP term
- term → factor (+/- factor)
- factor → primary (\* or / primary)
- primary → NUMBER, IDENTIFIER, STRING, (...)

## **Desteklenen yapılar:**

- Değişken tanımı: var x = 5;
- Ekrana yazdırma: print("merhaba");
- Koşullu ifade: if (x > 3) { ... };
- Döngü: while (x < 10) { ... }

## **İşleyiş:**

- Her yapının grameri metotlara ayrılmıştır (örneğin: var\_declaration, if\_statement, expression)
- Uygun olmayan sözdizimleri için SyntaxError fırlatılır.

## **Operatör kontrolü:**

Sadece aşağıdaki operatörler geçerlidir:

+ - \* / == != < > <= >= == ++ --

Bunlardan farklı bir operatör kullanılırsa hata fırlatılır.

## Örnek Girdi

```
code = '''
var a = 3 + 4 * (2 - 1);
if (a >= 7) {
    print("Merhaba Dünya");
};
'''
```

## Çıktı

```
✓ Matched: KEYWORD -> var
✓ Matched: IDENTIFIER -> a
✓ Matched: OPERATOR -> =
✓ Matched: NUMBER -> 3
✓ Matched: OPERATOR -> +
✓ Matched: NUMBER -> 4
✓ Matched: OPERATOR -> *
✓ Matched: DELIMITER -> (
✓ Matched: NUMBER -> 2
✓ Matched: OPERATOR -> -
✓ Matched: NUMBER -> 1
✓ Matched: DELIMITER -> )
✓ Matched: DELIMITER -> ;
✓ Matched: KEYWORD -> if
✓ Matched: DELIMITER -> (
✓ Matched: IDENTIFIER -> a
✓ Matched: OPERATOR -> >=
✓ Matched: NUMBER -> 7
✓ Matched: DELIMITER -> )
✓ Matched: DELIMITER -> {
✓ Matched: KEYWORD -> print
✓ Matched: DELIMITER -> (
✓ Matched: STRING -> Merhaba Dünya
✓ Matched: DELIMITER -> )
✓ Matched: DELIMITER -> ;
✓ Matched: DELIMITER -> }
✓ Matched: DELIMITER -> ;
```

## **gui.py**

Tkinter ile yapılmış grafiksel kullanıcı arayüzüdür. Kullanıcı buraya kodlarını yazar.



### **CodeEditorGUI sınıfı**

#### **Ana bileşenler:**

- text\_area: Kod yazılan alan
- error\_label: Hata/başarı mesajı
- token\_colors: Token tiplerine göre renklendirme (örneğin KEYWORD → blue)

#### **on\_text\_change**

Her tuş bırakıldığında:

1. lexer ile kod tokenlara ayrılır.
2. parser ile sözdizimi kontrol edilir.
3. Hata yoksa:  “Sözdizimi doğru”
4. Hata varsa:  Hata mesajı gösterilir.

#### **highlight\_tokens\_no\_pos**

Token'ları satır satır tarar ve tanınan sözcükleri renklendirir.

#### **Özetle:**

- Kod yazımı sırasında KeyRelease ile her tuş bırakıldığında analiz yapılır.
- Token türlerine göre renkli sözdizimi vurgulama yapılır.
- Hata varsa kırmızı uyarı etiketi gösterilir.



Gerçek Zamanlı Sözdizimi Denetleyici

```
var sayi=10;  
if(sayi>5)  
{  
print("sayi 5'ten büyük");  
}
```



Sözdizimi doğru.