



# BURSA TEKNİK ÜNİVERSİTESİ

Bursa Teknik Üniversitesi  
Bilgisayar Mühendisliği

## BLM0238 Programlama Dilleri

<b>Ad Soyad</b>	Edem Makhsudov
<b>Bölüm</b>	Bilgisayar Mühendisliği
<b>Öğrenci No</b>	22360859373
<b>Proje Konusu</b>	Real-Time Grammar-Based Syntax Highlighter with GUI
<b>Programlama Dili</b>	Python
<b>Hedef Dili</b>	TurkScript

# İÇİNDEKİLER

1. Dil ve Gramer Seçimi (Language and Grammar Choice)
2. Sözdizimi Analiz Süreci (Syntax Analysis Process)
3. Leksikal Analiz Detayları (Lexical Analysis Details)
4. Ayırıştırma Metodolojisi (Parsing Methodology)
5. Vurgulama Şeması (Highlighting Scheme)
6. GUI Implementasyonu (GUI Implementation)
7. Sonuçlar ve değerlendirme

# 1. DİL VE GRAMER SEÇİMİ

## 1.1 Programlama Dili Seçimi

Bu projede **TurkScript** adında özgün bir programlama dili tasarlanmıştır. Bu seçimin sebepleri:

- **Türkçe Anahtar Kelimeler:** Türkçe anahtar kelimeler ile daha anlaşılır
- **Orijinallik:** Mevcut dillerin kopyası olmayan özgün bir yaklaşım
- **Kontrol:** Grammar ve syntax kurallarının tam kontrolü
- **Demonstrasyon:** Compiler tasarım prensiplerinin net gösterimi

## 1.2 TurkScript Gramer Yapısı

### 1.2.1 Anahtar Kelimeler

Veri Tipleri: sayı, metin, mantık, liste  
Kontrol Yapıları: eger, yoksa, yoksa\_eger, dongu, iken, dur, devam  
Fonksiyonlar: fonksiyon, don, cagir  
Değişkenler: degisken, sabit  
Mantıksal: ve, veya, degil, dogru, yanlis  
Giriş/Çıkış: yazdir, oku

### 1.2.2 Formal Grammar (BNF Notasyonu):

```
<program> ::= <statement_list>
<statement_list> ::= <statement> | <statement> <statement_list>
<statement> ::= <var_declaration> | <function_declaration> |
               <if_statement> | <loop_statement> |
               <print_statement> | <assignment> | <function_call>

<var_declaration> ::= "degisken" [<type>] <identifier> "=" <expression>
";"
<type> ::= "sayi" | "metin" | "mantik" | "liste"
<function_declaration> ::= "fonksiyon" <identifier> "(" <param_list>
")" "{" <statement_list> "}"
<if_statement> ::= "eger" "(" <expression> ")" "{" <statement_list> "}"
["yoksa" "{" <statement_list> "}"]
<loop_statement> ::= ("dongu" | "iken") "(" <expression> ")" "{"
<statement_list> "}"
<print_statement> ::= "yazdir" "(" <expression> ")" ";"
<assignment> ::= <identifier> "=" <expression> ";"
<function_call> ::= ["cagir"] <identifier> "(" <arg_list> ")" ";"
```

```

<expression> ::= <logical_expression>
<logical_expression> ::= <comparison_expression>
                        [ ("ve" | "veya") <comparison_expression> ] *
<comparison_expression> ::= <arithmetic_expression>
                        [ ("==" | "!=" | "<" | ">" | "<=" | ">=")
<arithmetic_expression> ] *
<arithmetic_expression> ::= <term> [ ("+" | "-") <term> ] *
<term> ::= <factor> [ ("*" | "/" | "%") <factor> ] *
<factor> ::= <number> | <string> | <identifier> | <boolean> |
            "(" <expression> ")" | <function_call>

```

### 1.2.3 Operatörler

```

Aritmetik: +, -, *, /, %
Karşılaştırma: ==, !=, <, >, <=, >=
Atama: =
Mantıksal: &&, ||, !

```

## 2. SÖZDİZİMİ ANALİZ SÜRECİ

### 2.1 Genel Analiz Akışı

Kaynak Kod → Leksikal Analiz → Token Stream → Syntax Analiz → Parse Tree → Hata Kontrolü

### 2.2 İki Aşamalı Analiz Yapısı

#### 2.2.1 Leksikal Analiz Aşaması

- Kaynak kodun karakterlere ayrıştırılması
- Token'lara dönüştürme işlemi
- Leksikal hataların tespit edilmesi

#### 2.2.2 Syntax Analiz Aşaması

- Token stream'in grammar kurallarına göre kontrol edilmesi
- Parse tree oluşturulması
- Syntax hatalarının raporlanması

### 2.3 Real-Time Analiz Mekanizması

```
def syntax_highlighting_uygula(self):  
    # 1. Mevcut metni al  
    metin = self.metin_editor.get('1.0', tk.END)  
  
    # 2. Tokenize et  
    tokenlar = self.lexer.tokenize(metin)  
  
    # 3. Parse et  
    self.parser = TurkScriptParser(tokenlar)  
    self.parser.parse()  
  
    # 4. Hataları göster  
    self.hatalari_goster()  
  
    # 5. Syntax highlighting uygula  
    for token in tokenlar:  
        # Renk uygulama işlemi
```

## 3. LEKSİKAL ANALİZ DETAYLARI

### 3.1 Seçilen Metod: State Diagram & Program Implementation

Bu projede **State Diagram & Program Implementation** yaklaşımı seçilmiştir çünkü:

- **Esneklik:** Karmaşık token pattern'lerini destekler
- **Performans:** Direct implementation ile hızlı çalışma
- **Kontrol:** Her state transition'ın manuel kontrolü
- **Debugging:** Kolay hata ayıklama imkanı

### 3.2 Token Türleri ve Tanımlama

```
class TokenTuru:
    ANAHTAR_KELIME = "ANAHTAR_KELIME"      # eger, dongu, fonksiyon
    TANIMLAYICI = "TANIMLAYICI"             # değişken adları
    SAYI = "SAYI"                           # 123, 45.67
    METIN = "METIN"                         # "hello", 'world'
    OPERATOR = "OPERATOR"                   # +, -, ==, !=
    AYIRICI = "AYIRICI"                     # (, ), {, }, ;
    YORUM = "YORUM"                         # // yorum satırı
    BOSLUK = "BOSLUK"                       # \n, \t
    HATA = "HATA"                           # tanınmayan karakterler
    EOF = "EOF"                             # dosya sonu
```

### 3.3 Lexer State Diagram

```
[Başlangıç] → [Karakter Okuma]
↓
├─ Harf/_ → [Identifier/Keyword State]
├─ Rakam → [Number State]
├─ "/" → [String State]
├─ Operator → [Operator State]
├─ Separator → [Separator State]
├─ // → [Comment State]
├─ Whitespace → [Skip State]
└─ Diğer → [Error State]
```

## 3.4 Lexer Algoritması

### 3.4.1 Ana Tokenization Döngüsü

```
def tokenize(self, kaynak_kod=None):  
    ...  
    while True:  
        token = self.sonraki_token()  
        self.tokenlar.append(token)  
        if token.tur == TokenTuru.EOF:  
            break  
    return self.tokenlar
```

### 3.4.2 Sayı Tanıma Algoritması

```
def sayi_oku(self):  
    sayi_str = ""  
    # Tam sayı kısmı  
    while self.mevcut_karakter().isdigit():  
        sayi_str += self.mevcut_karakter()  
        self.ileri_git()  
  
    # Ondalık kısım kontrolü  
    if self.mevcut_karakter() == '.' and self.sonraki_karakter().isdigit():  
        sayi_str += self.mevcut_karakter()  
        self.ileri_git()  
        while self.mevcut_karakter().isdigit():  
            sayi_str += self.mevcut_karakter()  
            self.ileri_git()  
  
    return Token(TokenTuru.SAYI, sayi_str, self.satir, baslangic_sutun)
```

## 3.5 Hata Yönetimi

### 3.5.1 Leksikal Hatalar

- **Bilinmeyen karakterler:** Alfabet dışındaki semboller
- **Kapatılmamış string'ler:** Quote işareti eksik stringler

### 3.5.2 Hata Raporlama

```
def hata_rapor_et(self, mesaj):  
    hata_mesaji = f"{mesaj}: Satır {self.satir}, Sütun {self.sutun}"  
    self.hatalar.append(hata_mesaji)
```

## 4. AYRIŞTIRMA METODOLOJİSİ

### 4.1 Top-Down Parser Seçimi

**Top-Down (Yukarıdan Aşağı) Parser** seçilmesinin nedenleri:

- **Sadelik:** Anlaşılması ve implement edilmesi kolay
- **Predictive Parsing:** LL(1) grammar için uygun
- **Left-to-Right:** Doğal okuma sırası
- **Error Recovery:** Hata bulma ve raporlama kolaylığı

### 4.2 Recursive Descent Implementation

#### 4.2.1 Parser Yapısı

```
class TurkScriptParser:
    def __init__(self, tokenlar):
        self.tokenlar = tokenlar
        self.mevcut_indeks = 0
        self.hatalar = []

    def parse(self):
        self.parse_program()
```

#### 4.2.2 Grammar Rule Implementation

```
def parse_degisken_tanimla(self):
    # degisken [tip] isim = deger;
    self.bekle(TokenTuru.ANAHTAR_KELIME, 'degisken')

    # Opsiyonel tip kontrolü
    if self.mevcut_token().deger in ['sayi', 'metin', 'mantik', 'liste']:
        self.bekle(TokenTuru.ANAHTAR_KELIME)

    self.bekle(TokenTuru.TANIMLAYICI)      # değişken adı
    self.bekle(TokenTuru.OPERATOR, '=')    # atama operatörü
    self.parse_ifade()                     # değer ifadesi
    self.bekle(TokenTuru.AYIRICI, ';')     # statement sonu
```



## 4.3 Expression Parsing (İfade Ayırıştırma)

### 4.3.1 Operator Precedence Hierarchy

1. Mantıksal (ve, veya) - En düşük öncelik
2. Karşılaştırma (==, !=, <, >, <=, >=)
3. Aritmetik (+ -)
4. Çarpma/Bölme (\*, /, %)
5. Faktörler (sayılar, değişkenler, parantezler) - En yüksek öncelik

### 4.3.2 Left-Associative Parsing

```
def parse_aritmetik_ifade(self):
    self.parse_terim()

    while (self.mevcut_token().tur == TokenTuru.OPERATOR and
           self.mevcut_token().deger in ['+', '-']):
        self.sonraki_token()
        self.parse_terim()
```

## 4.4 Error Recovery Stratejisi

### 4.4.1 Panic Mode Recovery

```
def bekle(self, beklenen_tur, beklenen_deger=None):
    token = self.mevcut_token()

    if token.tur != beklenen_tur:
        hata_mesaji = f"Beklenen: {beklenen_tur}, Bulunan: {token.tur}"
        self.hatalar.append({
            'mesaj': hata_mesaji,
            'satir': token.satir,
            'sutun': token.sutun
        })
        raise ParseHatasi(hata_mesaji, token.satir, token.sutun)
```

## 5. VURGULAMA ŞEMASI

### 5.1 Renk Kodlama Sistemi

```
RENK_HARITASI = {
    TokenTuru.ANAHTAR_KELIME: '#0000FF', # Mavi - Görünürlük için
    TokenTuru.TANIMLAYICI: '#000000', # Siyah - Neutral
    TokenTuru.SAYI: '#FF6600', # Turuncu - Sayısal değerler
    TokenTuru.METIN: '#008000', # Yeşil - String literals
    TokenTuru.OPERATOR: '#800080', # Mor - Operators
    TokenTuru.AYIRICI: '#000000', # Siyah - Structural elements
    TokenTuru.YORUM: '#808080', # Gri - Comments (subtle)
    TokenTuru.HATA: '#FF0000', # Kırmızı - Hatalar
}
```

### 5.2 Real-Time Highlighting Algoritması

#### 5.2.1 Debounce Mechanism

```
def metin_degisti(self, event=None):
    self.son_guncelleme = time.time()
    self.root.after(self.guncelleme_gecikmesi, self.gec_guncelleme)

def gec_guncelleme(self):
    if time.time() - self.son_guncelleme >= self.guncelleme_gecikmesi /
1000:
        self.syntax_highlighting_uygula()
```

#### 5.2.2 Position Calculation

```
def pozisyon_hesapla(self, metin, char_pozisyon):
    satir = 1
    sutun = 0

    for i, char in enumerate(metin[:char_pozisyon]):
        if char == '\n':
            satir += 1
            sutun = 0
        else:
            sutun += 1

    return f"{satir}.{sutun}"
```

## 6. GUI İMPLEMENTASYONU

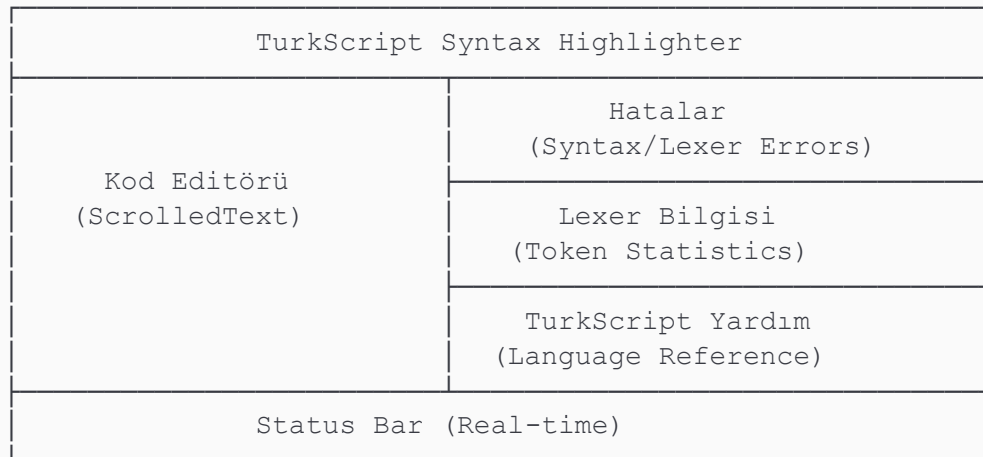
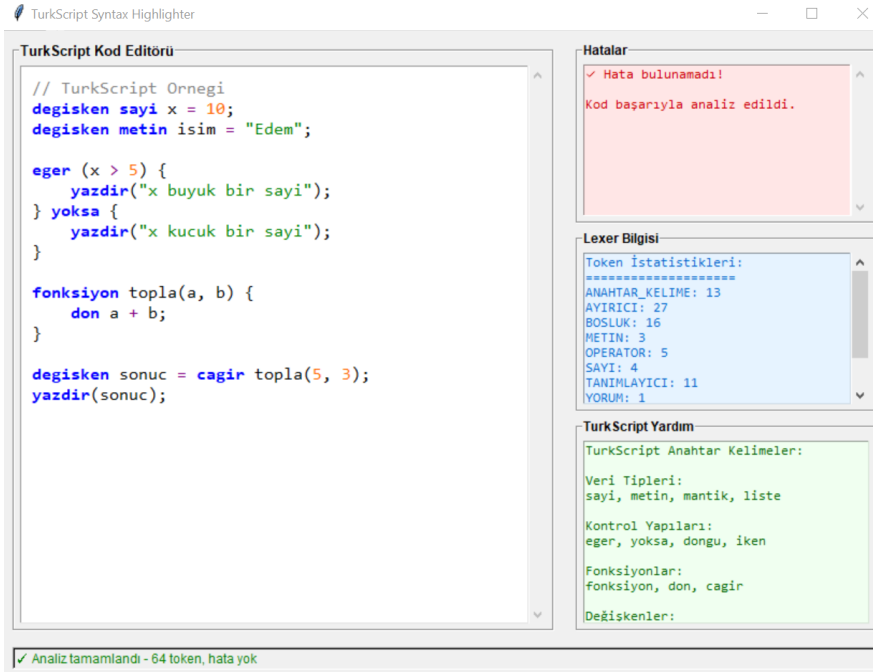
### 6.1 Tkinter Framework Seçimi

**Tkinter** seçilmesinin nedenleri:

- **Native Python support:** Ek kütüphane gerektirmez
- **Cross-platform:** Windows, Linux, macOS uyumluluğu
- **Zengin widget'lar:** ScrolledText, LabelFrame, Event handling.
- **Real-time updates:** Dinamik arayüz güncellemeleri için ideal

### 6.2 UI Architecture

#### 6.2.1 Responsive Layout Yapısı



## 6.2.2 Ana Bileşenler

- **Sol Panel:** Kod editörü (ana çalışma alanı)
- **Sağ Panel:** Üç bilgi paneli (hata, lexer, yardım)
- **Alt Panel:** Dinamik durum çubuğu

## 6.3 Event Handling ve Real-Time Analiz

### 6.3.1 Dinamik Güncelleme Mekanizması

```
def metin_degisti(self, event=None):  
    self.status_label.config(text="⌚ Analiz ediliyor...", fg='orange')  
    self.son_guncelleme = time.time()  
    self.root.after(self.guncelleme_gecikmesi, self.gec_guncelleme)
```

### 6.3.2 Debounce Stratejisi

- **300ms gecikme:** Gereksiz analiz işlemlerini önler
- **Performance optimizasyonu:** Hızlı yazma sırasında lag önleme
- **User experience:** Akıcı editör deneyimi

## 6.4 Akıllı Status Bar

```
# Duruma göre dinamik mesajlar  
if toplam_token == 0:  
    self.status_label.config(text="📄 Boş doküman", fg='gray')  
elif toplam_hata == 0:  
    self.status_label.config(  
        text=f"✓ Analiz tamamlandı - {toplam_token} token, hata yok",  
        fg='green'  
    )  
else:  
    self.status_label.config(  
        text=f"⚠ {toplam_hata} hata bulundu - {toplam_token} token ana-  
liz edildi",  
        fg='red'  
    )
```

## 7. SONUÇLAR VE DEĞERLENDİRME

Bu projede aşağıdaki hedefler başarıyla gerçekleştirilmiştir:

- **TurkScript dili:** Türkçe anahtar kelimelerle özgün programlama dili
- **Leksikal analizci:** 10 farklı token türünü tanıyan lexer
- **Syntax analizci:** Top-down parser ile grammar kontrolü
- **Real-time highlighting:** 300ms içinde canlı renklendirme
- **GUI arayüzü:** Tkinter ile kullanıcı dostu interface
- **Hata yönetimi:** Lexical ve syntax hatalarının gösterimi

TurkScript Syntax Highlighter başarıyla tamamlanmış ve çalışır durumda bir syntax highlighter'dır.