***Gerçek Zamanlı Biçimbilgisel Temelli Sözdizimi Vurgulama Arayüzü***

***Proje Başlığı***

Gerçek Zamanlı Biçimbilgisel Temelli Sözdizimi Vurgulama Arayüzü

***Yazar***

Rima Farah Eleuch  
Öğrenci Numarası: 21360859216

***Kullanılan Programlama Dili ve Araçlar***

* **Programlama Dili:** Python 3.11
* **Geliştirme Ortamı:** PyCharm, Jupyter Notebook (ilk taslaklar için kullandım)
* **GUI Kütüphanesi:** Tkinter (Python standart kütüphanesi)
* **Test Kütüphanesi:** unittest

***Dil ve Gramer Seçimi***

Bu projede Python’un yazım tarzından ilham alarak daha basit bir programlama dili oluşturdum. Desteklenen yapılar:

* Atama ifadeleri
* Aritmetik ifadeler
* Mantıksal ifadeler
* “if, elif, else, while” blokları
* “print()” fonksiyonu
* Blokları kapatmak için “end” kullanımı
* “#” ile başlayan yorum satırları

***Örnek:***

x = 3 + 4 \* (2 - 1)

if x == 7:

print("Doğru")

else:

print("Yanlış")

end

Gramer, manuel olarak analiz edilmesi kolay olacak şekilde tasarladım ancak temel sözdizim analizini gerçekleştirmek için yeterince zengin.

***Sözdizimi (Syntax) Analizi Süreci***

**Genel Yaklaşım:**

**Üstten-Aşağı (Top-Down) Özyinelemeli (Recursive Descent)** analiz yöntemi tercih ettim. Bu yöntem, bir çözümleyici (parser) üreticiye ihtiyaç duymadan uygulanabilir.

***Gramer Taslağı:***

Program -> Statement\*

Statement -> Assignment | IfStmt | WhileStmt | PrintStmt | COMMENT

Assignment -> IDENTIFIER '=' Expression

IfStmt -> 'if' Expression ':' Block ( 'elif' Expression ':' Block )\* ( 'else' ':' Block )? 'end'

WhileStmt -> 'while' Expression ':' Block 'end'

PrintStmt -> 'print' '(' Expression (',' Expression)\* ')'

Block -> Statement\*

Expression -> LogicalOr

...

Parser hataları algılayabilir ve ayrıntılı hata mesajları verir. Eksik end ifadeleri ve yersiz elif/else kullanımları tespit edilir.

***Sözcüksel (Lexical) Analiz Detayları***

***Kullanılan Yaklaşım:***

**Durum diyagramı ve programatik kurallar (State Diagram And Program Implementation)** temelinde manuel olarak bir lexer (sözcüksel çözümleyici) yazdım. Dış kütüphaneler kullanılmadı.

***Token (Belirteç) Türleri:***

* KEYWORD: if, else, elif, while, end, print, true, false
* IDENTIFIER: değişken adları (x, toplam, vs.)
* NUMBER: tam sayı ve ondalık sayılar
* STRING: tırnak içerisinde string ifadeler
* SYMBOL: semboller ve operatörler (=, +, :, ==, ||, vb.)
* COMMENT: # ile başlayan yorumlar
* ERROR: hatalı string ifadeleri
* UNKNOWN: bilinmeyen karakterler

***Ele Alınan Uç Durumlar:***

* Kaçış karakterleri olan stringler
* Birden fazla nokta içeren sayılar (1.2.3)
* Yorumlar token olarak korunur ve vurgulanır

***Ayrıştırma (Parsing) Yöntemi***

***Yöntem:***

Üstten-aşağı özyinelemeli ayrıştırıcı **(Top-Down Recursive Descent Parser)**

***Yapı:***

* **parser\_module.py** dosyasında uygulandı
* Token listesi lexer'dan alınır
* Fonksiyonlar gramer kurallarını temsil eder: expr(), if\_stmt(), while\_stmt(), vb.

***Hata Yönetimi:***

* Beklenmeyen token'lar için hata mesajları listeye eklenir
* GUI'de yalnızca ilk 3 hata gösterilir

***Özel Durumlar:***

* **elif** ve **else** yalnızca **if** bloğu içinde geçerlidir
* **end** ifadesi **if** ve **while** bloklarını sonlandırmak için zorunludur

***Vurgulama Şemasi***

Vurgulama tk.Text widget'ın etiketleme sistemini kullanarak yapılır.

***Token Türleri ve Renkleri:***

|  |  |
| --- | --- |
| Token Türü | Renk |
| KEYWORD | Siyah |
| IDENTIFIER | Mavi |
| NUMBER | Yeşil |
| STRING | Magenta |
| SYMBOL | Turuncu |
| COMMENT | Gri |
| ERROR | Kırmızı |
| UNKNOWN | Pembe |

***Gerçek Zamanlı Özellik:***

* Her tuş bırakılışında içerik yeniden token'lara ayrılır
* Tüm metin tekrar taranır ve renklendirilir

***Performans:***

* Kısa kodlar için yeterli hızda çalışıyor (~100 satır)
* Daha büyük dosyalar için optimize edilmemiştir, bu proje için kabul edilebilir

***Grafik Arayüz (GUI) Uygulaması***

**GUI tkinter** kullanılarak geliştirdim:

* Kod yazma alanı ve kaydırma çubuğu
* Gerçek zamanlı vurgulama
* Düğmeler:
  + "Çözümle": Parser'ı çalıştırır ve hataları gösterir
  + "Temizle": Alanı temizler
* Durum çubuğu: Başarı veya hata mesajlarını gösterir

***Dosya: gui.py***

Fonksiyonlar:

* setup\_ui(): Arayüz bileşenlerini kurar
* highlight\_code(): Token'a göre renk uygular
* parse\_code(): Parser'ı çalıştırır, hataları gösterir

***Sınırlar:***

* Dosya kaydetme/yükleme yok
* Otomatik girinti veya sekme desteği yok

***Testler***

Testler şu dosyalarda yer alıyor:

* test\_tokenizer.py: Lexer için çeşitli durumlar
* test\_parser.py: Doğru/yanlış sözdizimi, iç içe bloklar, eksik end vb.

Python unittest yapısı kullanılarak otomatik testler gerçekleştirdim.

***Bilinen Sorunlar / Gelecek Geliştirmeler***

* Uzun girdilerde vurgulama performansı düşebilir
* İfade hatası kurtarma geliştirilebilir
* Dosya aç/kaydet fonksiyonu eklenebilir
* Otomatik girintileme / tamamlayıcı eklenebilir

***Sonuç***

Bu proje, tamamen sıfırdan geliştirilmiş gerçek zamanlı bir sözdizimi vurgulama uygulamasını başarıyla sunmaktadır. Hiçbir harici kütüphane kullanılmadım, temel ayrıştırma ve vurgulama ilkeleri uyguladım.