Python ile Sıfırdan Gerçek Zamanlı Sözdizimi Renklendirici Nasıl Yapılır?

Giriş

Her gün kullandığımız kod editörlerinin, yazdığımız kodu sihirli bir şekilde anlayıp renklendirmesi nasıl mümkün oluyor? `if` komutları neden mavi, metinler neden pembe görünür? Bu yazıda, bu "sihrin" arkasındaki mantığı inceleyecek ve Pygments gibi hazır kütüphaneler kullanmadan, sıfırdan geliştirdiğimiz gerçek zamanlı bir sözdizimi renklendiricinin mimarisini ve çalışma prensiplerini adım adım açıklayacağız.

Projemiz, bir dilin yapısını iki temel aşamada analiz eder: **sözcüksel analiz (lexical analysis)** ve **sözdizimsel analiz (parsing)**.

Projenin Üç Temel Bileşeni

Projemiz üç ana modülden oluşmaktadır:

1. **lexer.py (Sözcüksel Analizci):** Kaynak kodu anlamlı "kelimelere" (token) ayıran bileşen.
2. **parser.py (Sözdizimsel Analizci/Ayrıştırıcı):** Bu kelimelerin bir araya gelerek dilin "cümle kurallarına" yani gramere uygunluğunu denetleyen bileşen.
3. **gui.py (Grafiksel Kullanıcı Arayüzü):** Tüm bu süreçleri bir araya getiren ve sonucu kullanıcıya anlık olarak sunan arayüz.

Adım 1: Sözcüksel Analiz - Kodu Anlamlı Parçalara Ayırmak

Her şey, yazdığımız kodun ham bir metin olmasıyla başlar. Lexer'ın görevi, bu metni okuyarak dilimiz için anlamlı olan "token" adı verilen parçacıklara ayırmaktır.

Projemizdeki Lexer, kaynak kodu karakter karakter işler. Bir karakter dizisiyle karşılaştığında bunu bir IDENTIFIER (tanımlayıcı) olarak etiketler ve bunun bir anahtar kelime (`if`, `while` vb.) olup olmadığını kontrol eder. Rakamlarla karşılaştığında bir NUMBER, tırnak işaretleri arasında ise bir STRING_LITERAL token'ı oluşturur.

Bu aşama, temel renklendirmenin ilk adımıdır ve her bir "kelimenin" ne olduğunu anlamamızı sağlar.

Adım 2: Sözdizimsel Analiz - Gramer Kurallarını Denetlemek

Lexer'dan elde ettiğimiz token dizisi, kelimelerin tek tek anlamlı olduğunu söyler, ancak cümlenin bir bütün olarak doğru kurulup kurulmadığını söylemez. İşte bu noktada Parser devreye girer.

Parser'ımız, **"Top-Down Recursive Descent"** (Yukarıdan Aşağıya Özyinelemeli İniş) adı verilen bir yöntem kullanır. Bu yöntemde, dilin gramerindeki her bir kural (örneğin bir `if`

ifadesi veya bir değişken ataması) için bir fonksiyon bulunur. Parser, token dizisini baştan sona okuyarak bu kurallara uygun bir yapı olup olmadığını denetler.

Örneğin, bir `if` token'ı gördüğünde, ardından (, bir koşul ifadesi,) ve bir kod bloğu ({...}) gelmesini bekler. Eğer bu yapıya uymayan bir durumla karşılaşırsa, bir ParseError (Ayrıştırma Hatası) fırlatır. Bu hata, arayüzün kullanıcıya tam olarak nerede yanlış yaptığını göstermesini sağlar.

Adım 3: Arayüz ve Gerçek Zamanlı Etkileşim

Projenin kalbi olan gui.py, tkinter kullanılarak oluşturulmuş basit ama güçlü bir arayüzdür.

- **Gerçek Zamanlı Tetikleme:** Kullanıcı metin alanına her karakter girdiğinde (<KeyRelease> olayı), renklendirme ve analiz süreci yeniden başlar.
- **Renklendirme:** highlight fonksiyonu önce lexer'ı çağırır. Gelen token listesindeki her bir token'ın tipine göre, ModernTheme sınıfında tanımlanan renkleri uygular. Örneğin, TokenType.IF mavi, TokenType.STRING_LITERAL pembe renkte gösterilir.
- **Hata Gösterimi:** highlight fonksiyonu, renklendirme sonrası parser'ı bir `try...except` bloğu içinde çalıştırır. Bir ParseError yakalanırsa, arayüzün altındaki etiket güncellenir ve hatalı token kırmızı bir arka planla vurgulanır.

Sonuç

Bu proje, bir programlama dilinin temel analiz süreçlerini sıfırdan inşa etmenin harika bir yoludur. Lexer ve Parser arasındaki bu iki aşamalı yapı, modern derleyicilerin ve yorumlayıcıların temel çalışma prensibini oluşturur. Kendi aracınızı geliştirmek, sadece keyifli bir meydan okuma olmakla kalmaz, aynı zamanda programlama dillerinin ardındaki mantığı anlamak için de paha biçilmez bir deneyim sunar.