DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Temel Python Komutları
ÜNİTE NO 1
YAZAR Dr. RAZİYE KILIÇ SARIGÜL

TEMEL PYTHON KOMUTLARI

Bu ünite, Python programlama diline başlangıç amacıyla hazırlanmıştır ve öğrencinin temel kavramları hızlı biçimde görmesini hedefler. Python'un okunabilir sözdizimi, açık kaynak yapısı ve platform bağımsızlığı sayesinde eğitimden endüstriyel uygulamalara uzanan geniş bir kullanım alanı vardır. Ünite boyunca, Python'un neden yaygın tercih edildiği, nasıl çalıştırıldığı ve temel programlama bileşenlerinin (veri türleri, operatörler, değişkenler, girdi–çıktı ve tür dönüşümleri) nasıl kullanıldığı bütüncül bir çerçevede ele alınır.

PYTHON PROGRAMLAMA: TEMELLER VE ARAÇLAR

Programlama dilleri, yazılan kodun bilgisayar tarafından nasıl çalıştırıldığına göre genellikle derlenen ve yorumlanan diller olarak açıklanır. Derlenen dillerde kaynak kod önce bir derleyici tarafından makine komutlarına çevrilir ve ortaya çalıştırılabilir bir çıktı çıkar. Bu yaklaşım genellikle daha yüksek çalışma hızı sunarken, her değişiklikten sonra yeniden derleme gerektirebilir ve çıktının işletim sistemine veya işlemciye göre değişmesi taşınabilirlik planlamasını zorlaştırabilir. Yorumlanan dillerde ise kod, program çalıştırılırken yorumlayıcı tarafından adım adım işlenir. Bu yapı, deneme ve geliştirme sürecini kolaylaştırır ancak bazı senaryolarda derlenen dillere göre daha düşük performans görülebilir. Python bu çerçevede yorumlanan diller arasında ele alınır.

Python'un pratik değerini artıran unsurlardan biri de paket ve kütüphane ekosistemidir. Çekirdek dil temel ihtiyaçları karşılarken, daha kapsamlı çalışmalar için üçüncü taraf paketler kullanılabilir. Bilimsel hesaplama ve veri analizi bağlamında NumPy, SciPy, Matplotlib ve Pandas gibi paketler örnek verilir. Paket kurulumunda pip gibi araçlar kullanılabilir. Ayrıca Anaconda gibi dağıtımlar, çok sayıda aracı tek kurulumla sunarak düzenli bir başlangıç sağlar ve farklı projeler için farklı paket sürümlerini "ortam" yönetimiyle ayırmayı kolaylaştırır. Google Colab ise tarayıcı üzerinden çalışan, kurulum gerektirmeyen bulut tabanlı bir Jupyter Notebook ortamı sunar ve özellikle eğitim ile makine öğrenmesi çalışmalarında yaygındır. Bununla birlikte Colab oturumlarının sürekliliği ve kaynak kısıtları değişken olabildiği için bu koşulların öğrenme ve uygulama sürecinde dikkate alınması gerekir.

PYTHON'UN ÇALIŞMA MANTIĞI

Python ile program geliştirme süreci temelde iki adımdan oluşur. İlk adımda geliştirici, komutları bir metin düzenleyicide yazar ve genellikle .py uzantılı bir dosyaya kaydeder. İkinci adımda bu dosya Python yorumlayıcısı tarafından yürütülür. Bilgisayarlar Python gibi insanın okuyabileceği düzeyde yazılmış dilleri doğrudan çalıştıramadığı için yorumlayıcı, kodu okuyup anlamlandırarak yürütme adımlarını oluşturur. CPython bu bağlamda en yaygın yorumlayıcı uygulamasıdır ve kullanıcı açısından temel nokta, "çalıştır" komutu verildiği anda kodun yorumlanarak sonucun üretilmesidir.

Kod çalıştırma biçimi iki yaygın kullanım üzerinden açıklanır. Etkileşimli kullanımda komutlar satır satır girilir ve sonuçlar anında görülür. Bu yaklaşım özellikle kısa denemeler ve sözdizimi öğrenimi için işlevseldir. Dosya çalıştırma yaklaşımında ise Python komutları bir dosyada tutulur ve yorumlayıcı dosyayı baştan sona yürütür. Bu yöntem, kodun saklanması, tekrar çalıştırılması ve daha düzenli geliştirilmesi açısından avantaj sağlar. Her iki durumda da kodu yürüten bileşen editör değil, Python yorumlayıcısıdır. Ayrıca kodda yazım hatası olduğunda Python çalışmayı durdurur ve hatanın görüldüğü satıra ilişkin mesaj üretir. Bu mesajlar öğrenme sürecinde geri bildirim işlevi görür.

TEMEL VERİ TÜRLERİ

Programlamada veri türü, bilginin bilgisayarda nasıl temsil edildiğini ve bu bilgi üzerinde hangi işlemlerin yapılabileceğini belirler. Python'da temel veri türlerini kavramak, programın doğru çalışması ve beklenen çıktının üretilmesi için kritik bir başlangıçtır. Ünite, dört temel veri tipine odaklanır: tam sayı (int), ondalıklı sayı (float), metin (str) ve mantıksal değer (bool). Tam sayılar sayma ve temel aritmetik işlemlerde kullanılır. Ondalıklı sayılar ölçme ve oran hesaplarında yaygındır. Metinler tırnak içinde yazılır ve sayıya benzese bile tırnak varsa metin olarak değerlendirilir. Mantıksal tür ise True ve False değerleri üzerinden koşul değerlendirmelerinde

kullanılır.

Bir değerin türünü tanımak için type() fonksiyonu kullanılır. Bu işlev, özellikle metin ile sayısal değerlerin ayırt edilmesinde öğreticidir. Veri türü, program davranışını doğrudan etkiler. Örneğin sayılarla + operatörü toplama yaparken, metinlerde + birleştirme anlamına gelir. Benzer biçimde kullanıcıdan alınan girdiler çoğunlukla metin türünde geldiği için sayısal işlem yapılacaksa tür dönüşümü gerektirir. Bu vurgu, ilerleyen başlıklarda ele alınacak girdi alma ve tür dönüştürme konularına kavramsal temel sağlar.

PYTHON'DA ARİTMETİK OPERATÖRLER VE ÖNCELİK KURALLARI

Operatörler, değerler üzerinde işlem yapmayı sağlayan semboller olarak programlamanın merkezinde yer alır. Python'da toplama, çıkarma, çarpma ve bölme gibi temel işlemlerin yanında üs alma (**), tam bölme (//) ve kalan bulma (%) operatörleri de bulunur. Bölme işlemi çoğu durumda ondalıklı sonuç ürettiği için sonuç türü float olabilir. Tam bölme, bölümün aşağı yuvarlanmış tam sayı kısmını verir. Kalan bulma ise bir sayının başka bir sayıya tam bölünüp bölünmediğini anlamada ve döngüsel kontrol gerektiren durumlarda işlevseldir.

Aynı ifadede birden fazla operatör bulunduğunda, Python işlemleri belirli bir öncelik sırasına göre değerlendirir. Genel kural olarak parantez içi önce hesaplanır. Ardından üs alma değerlendirilir. Sonrasında çarpma, bölme, tam bölme ve kalan operatörleri soldan sağa işlenir. En sonda toplama ve çıkarma yine soldan sağa değerlendirilir. İşlem sırasını açık biçimde kontrol etmek için parantez kullanılır. Parantez yalnızca okunabilirliği artırmaz, aynı zamanda programın ürettiği sonucu da değiştirir. Bu nedenle işlem önceliği, temel aritmetik ifadelerin doğru kurulması için zorunlu bir bilgidir.

DEĞİŞKENLER VE ATAMA KOMUTU

Değişkenler, program içinde bir değeri temsil eden isimlerdir ve veriyi daha sonra tekrar kullanmayı, güncellemeyi ve okunabilir kod yazmayı sağlar. Python'da değişken oluşturmanın temel yolu atamadır ve = sembolü matematiksel eşitlikten farklı olarak "sağdaki değeri soldaki isme ata" anlamında kullanılır. Bir değişkene yeni bir değer atandığında önceki değer korunmaz ve değişken artık en son atanan değeri temsil eder. Değeri gerçekten güncellemek için yapılan hesaplamanın sonucunun tekrar aynı değişkene atanması gerekir. Bu yaklaşım, programın durumunun nasıl değiştiğini anlamak açısından temel bir beceridir.

Değişken adlandırma, programın doğruluğunu doğrudan değiştirmese de kodun anlaşılabilirliği ve sürdürülebilirliği üzerinde belirleyicidir. Python'da değişken adları harf veya alt çizgi ile başlayabilir, sayı ile başlayamaz, boşluk içeremez ve yalnızca harf, sayı ve alt çizgi içerebilir. Ayrıca Python büyük küçük harfe duyarlıdır ve ayrılmış sözcükler değişken adı olarak kullanılamaz. Birden fazla kelimeden oluşan adlarda alt çizgi ile ayırma yaklaşımı yaygındır. Anlamlı ve tutarlı adlar, kodu okuyan kişinin değişkenin neyi temsil ettiğini hızlıca kavramasına katkı sağlar.

TEMEL GİRDİ VE ÇIKTI İŞLEMLERİ

Bir programın kullanıcıyla en temel etkileşimi, kullanıcıdan girdi almak ve ekrana çıktı vermek üzerinden kurulur. Python'da girdi almak için input() fonksiyonu kullanılır. Program bu satıra geldiğinde kullanıcıdan değer girmesi beklenir ve Enter'a basılmadan süreç tamamlanmaz. Bu mekanizma, etkileşimli kabukta daha görünür biçimde deneyimlenir. Kritik nokta, input() ile alınan değerin varsayılan olarak metin türünde gelmesidir. Bu nedenle sayısal işlem yapılacaksa girilen metni uygun sayısal türe dönüştürmek gerekir.

Çıktı üretmek için print() fonksiyonu kullanılır ve parantez içindeki değer ekrana yazdırılır. Sayısal bir değeri metinle birlikte yazdırmanın temel yollarından biri, print() içinde virgülle ayırmaktır. Metin birleştirme yapılacaksa sayısal değerin str() ile metne dönüştürülmesi gerekir. Ayrıca metin uzunluğunu bulmak için len() fonksiyonu kullanılabilir. Ünite, yorum satırlarının # ile yazıldığını ve Python tarafından çalıştırılmadığını da vurgular. Yorumlar, kodun amacını açıklamak ve gerektiğinde bazı satırları geçici olarak devre dışı bırakmak için öğrenme sürecinde işlevsel bir araçtır.

TÜR DÖNÜŞÜMLERİ

Tür dönüşümü, bir değerin veri tipini başka bir tipe çevirme işlemidir ve özellikle kullanıcı girdisiyle çalışırken sık ihtiyaç duyulur. Bunun temel nedeni, input() ile alınan değerlerin metin türünde gelmesidir. Python'da başlangıç düzeyinde üç dönüşüm işlevi öne çıkar: int(), float() ve str(). int() metni tam sayıya, float() metni ondalıklı sayıya, str() ise sayısal bir değeri metne dönüştürür. Bu dönüşümler sayesinde kullanıcıdan alınan metinsel veriler sayısal işlemlerde kullanılabilir ve sayısal

sonuçlar metinle birlikte anlamlı çıktılara dönüştürülebilir.

Ünite, dönüşüm sürecini somut örneklerle pekiştirir. Tam sayı beklenen bir durumda kullanıcıdan alınan metin int() ile dönüştürülerek aritmetik işlemler yapılabilir. Ondalıklı değerlerde float() tercih edilir. Metin birleştirmede ise sayı doğrudan metne eklenemeyeceği için önce str() ile dönüşüm yapılır. Bu konu, veri türlerinin program davranışını nasıl etkilediğini yeniden görünür kılar ve öğrencinin hem hata yapma olasılığını azaltır hem de kodun amacına uygun çıktılar üretmesine destek olur.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Python'da Akış Kontrol
ÜNİTE NO 2
YAZAR Dr. ÖZGE ALBAYRAK ÜNAL

Giriş

Programlama dillerinde komutlar varsayılan olarak yukarıdan aşağıya doğru sıralı biçimde yürütülür. Bu yürütme modeli sıralı kontrol akışı olarak adlandırılır. Ancak gerçek yaşam problemleri çoğu zaman doğrusal değildir. Farklı koşullara göre farklı işlemlerin yapılması, bazı işlemlerin tekrar edilmesi veya belirli durumlarda sürecin durdurulması gerekebilir. Bu nedenle kontrol akışı yapıları, programın davranışını belirleyen temel mekanizmalardır.

Kontrol akışı; bir programın hangi satırlarının hangi koşullar altında çalıştırılacağını belirleyen yapıları ifade eder. Bu yapıların temel amacı, programı daha esnek, dinamik ve problem çözmeye uygun hâle getirmektir. Kontrol akışı yapılarının anlaşılabilmesi için öncelikle koşul kavramının mantıksal temellerinin ele alınması gerekmektedir. Bu kapsamda Boolean veri tipi, karşılaştırma operatörleri ve mantıksal operatörler incelenmekte ardından karar yapıları ve döngü mekanizmaları ile bu kavramların programlama pratiğindeki uygulamaları açıklanmaktadır.

Koşul Kavramının Mantıksal Temelleri

Kontrol akışı yapılarının anlaşılabilmesi için öncelikle koşul kavramının mantıksal temellerinin kavranması gerekmektedir. Programın farklı davranışlar sergileyebilmesi, belirli ifadelerin doğruluk değerine bağlıdır. Bu doğruluk değerlendirmesinin temelinde Boolean veri tipi yer alır.

Boolean veri tipi iki değer alır: True (Doğru) ve False (Yanlış). Ancak kontrol akışı yalnızca bu iki değerin bilinmesiyle sınırlı değildir. Bu değerlerin nasıl üretildiği ve nasıl bir araya getirildiği de önemlidir.

Boolean değerler çoğunlukla karşılaştırma operatörleri aracılığıyla elde edilir. İki değişkenin eşit olup olmadığı, bir değerin diğerinden büyük ya da küçük olup olmadığı gibi ilişkiler karşılaştırma operatörleri ile değerlendirilir ve sonuç Boolean türünde üretilir. Bu sonuç, programın hangi yönde ilerleyeceğini belirler.

Birden fazla koşulun birlikte değerlendirilmesi gerektiğinde mantıksal operatörler kullanılır. and, or ve not operatörleri sayesinde karmaşık koşul yapıları oluşturulabilir. Bu operatörler, algoritmaların çok boyutlu karar mekanizmalarını modelleyebilmesini sağlar. Böylece program yalnızca tek bir koşula değil, birden fazla değişkenin birlikte değerlendirilmesine dayalı olarak karar verebilir.

Karar Yapıları

Programın belirli bir koşula bağlı olarak farklı işlem yollarına ayrılması şartlı dallanma olarak adlandırılır. Python'da bu mekanizma if, if-else ve if-elif-else yapıları ile gerçekleştirilir.

if yapısı, tek yönlü bir karar mekanizmasıdır. Koşul doğru olduğunda ilgili kod bloğu çalıştırılır. Koşul yanlışsa program akışı normal seyrinde devam eder.

if-else yapısı, iki alternatifli durumlar için kullanılır. Koşul doğruysa bir işlem, yanlışsa başka bir işlem gerçekleştirilir. Bu yapı, karşılıklı iki durumun modellenmesinde etkilidir.

if-elif-else yapısı ise birden fazla koşulun sıralı biçimde değerlendirilmesini sağlar. Bu sayede çoklu karar mekanizmaları sistematik biçimde modellenebilir. Koşullar yukarıdan aşağıya doğru değerlendirilir ve ilk doğru koşulun bloğu çalıştırılır.

Döngü Yapıları

Bazı problemlerde belirli işlemlerin birden fazla kez tekrar edilmesi gerekir. Bu tekrar işlemleri döngü yapıları aracılığıyla gerçekleştirilir. Python'da iki temel döngü yapısı bulunmaktadır: while ve for.

while döngüsü

while döngüsü, koşul temelli yineleme sağlar. Koşul doğru olduğu sürece döngü çalışmaya devam eder; koşul yanlış olduğunda ise sonlanır. Bu yapı, tekrar sayısının önceden bilinmediği durumlar için uygundur.

Döngü her iterasyonda koşulu yeniden değerlendirir. Bu nedenle koşulun bir noktada yanlış olmasını sağlayacak bir güncelleme yapılmalıdır. Aksi takdirde döngü sonsuz hâle gelebilir. while yapısı özellikle kullanıcı girdisine bağlı veya dinamik süreçlerde etkili bir kontrol mekanizması sunar.

Döngü kontrol komutları

Döngülerin akışını daha esnek biçimde yönetmek için break ve continue komutları kullanılır.

break komutu, içinde bulunulan döngüyü tamamen sonlandırır ve program akışını döngü dışına yönlendirir. Belirli bir koşul gerçekleştiğinde tekrarın sürdürülmesinin gereksiz olduğu durumlarda kullanılır.

continue komutu ise döngüyü sonlandırmaz; yalnızca mevcut iterasyonda kalan işlemleri atlayarak bir sonraki yineleme adımına geçilmesini sağlar. Bu yapı, seçici kontrol mekanizması sunar ve döngü akışını kesintiye uğratmadan düzenler.

for döngüsü

for döngüsü, yinelenebilir veri yapıları üzerinde eleman bazlı yineleme yapılmasını sağlar. Belirli sayıda tekrar gerektiren ya da bir veri kümesinin tüm elemanları üzerinde işlem yapılması gereken durumlarda kullanılır.

Geleneksel programlama dillerinde sayaç temelli kullanılan for yapısı, Python'da iterable mantığına dayanmaktadır. Bu yaklaşım, veri yapılarının her bir öğesine sistematik ve okunabilir biçimde erişilmesini sağlar. Belirli sayıda tekrar gerektiren durumlarda range() fonksiyonu ile birlikte kullanılır.

range() Fonksiyonu

range() fonksiyonu, belirli bir aralıkta ardışık sayılar üretmek amacıyla kullanılan yerleşik bir fonksiyondur. Genellikle for döngüsü ile birlikte kullanılır.

FONKSİYONLAR VE PYTHON'DA UYGULANMASI

Programlama sürecinde geliştirilen yazılımlar genellikle birden fazla işlem ve görev içerir. Kullanıcıdan veri alma, hesaplama yapma, sonucu ekrana yazdırma veya belirli koşulları kontrol etme gibi işlemler aynı program içinde yer alabilir. Tüm bu işlemlerin tek bir ana blok içinde yazılması, kodun karmaşıklaşmasına ve yönetimin zorlaşmasına neden olur. Bu nedenle programlama dillerinde, kodu daha düzenli ve anlaşılır hâle getiren yapılar geliştirilmiştir. Fonksiyonlar bu yapıların başında gelmektedir.

Fonksiyonlar, belirli bir görevi yerine getirmek amacıyla oluşturulan bağımsız kod bloklarıdır. Tanımlandıktan sonra ihtiyaç duyulan yerde çağrılarak çalıştırılırlar. Bu yapı sayesinde program daha modüler bir hâl alır. Modülerlik, bir programın küçük ve anlamlı parçalara ayrılması anlamına gelir. Bu durum hem okunabilirliği artırır hem de bakım süreçlerini kolaylaştırır. Bir değişiklik yapılması gerektiğinde yalnızca ilgili fonksiyon güncellenir; programın diğer bölümlerine müdahale edilmesine gerek kalmaz.

Python programlama dilinde hem hazır fonksiyonlar hem de kullanıcı tarafından tanımlanan fonksiyonlar bulunmaktadır. print(), len() gibi fonksiyonlar hazır olarak sunulurken, programcı kendi ihtiyaçlarına göre yeni fonksiyonlar tanımlayabilir. Bu ünitede özellikle kullanıcı tanımlı fonksiyonlar ele alınmıştır.

Fonksiyon Tanımlama, Parametre ve Return Mekanizması

Python'da bir fonksiyon tanımlamak için def anahtar sözcüğü kullanılır. Fonksiyon adı yazıldıktan sonra parantez açılır ve gerekiyorsa parametreler belirtilir. Parantez kapatıldıktan sonra iki nokta üst üste işareti konur ve fonksiyon gövdesi girintili şekilde yazılır. Python'da girintileme yalnızca görsel bir düzen unsuru değil, aynı zamanda sözdizimin bir parçasıdır. Yanlış girintileme hataya neden olur.

Fonksiyonlar parametre alabilir ya da parametresiz olarak tanımlanabilir. Parametreler, fonksiyon tanımı sırasında parantez içinde belirtilen ve fonksiyonun dışarıdan veri almasını sağlayan değişkenlerdir. Fonksiyon çağrılırken bu parametrelere gönderilen gerçek değerlere ise argüman adı verilir. Parametre ve argüman kavramlarının doğru anlaşılması, fonksiyon kullanımının temelini oluşturur.

Bir fonksiyon tek parametre alabileceği gibi birden fazla parametre de alabilir. Birden fazla parametre kullanıldığında gönderilen değerlerin sırası önemlidir. Ancak anahtar kelimeli argüman kullanımı sayesinde parametre isimleri belirtilerek sıralama zorunluluğu ortadan kaldırılabilir. Bu yöntem kodun okunabilirliğini artırır.

Varsayılan parametre kavramı da fonksiyonların esnekliğini artıran önemli bir özelliktir. Fonksiyon tanımlanırken bazı parametrelere başlangıç değeri verilebilir. Fonksiyon çağrılırken bu parametreye değer gönderilmezse, varsayılan değer kullanılır. Ancak varsayılan parametrelerin zorunlu parametrelerden sonra ve parametre listesinin sonunda yer alması gerekir. Aksi hâlde sözdizimi hatası oluşur.

Fonksiyonların önemli özelliklerinden biri de değer döndürebilmeleridir. Bu işlem return anahtar sözcüğü ile gerçekleştirilir. Return ifadesi, fonksiyon içinde hesaplanan sonucu çağıran programa geri iletir. Return çalıştığında fonksiyonun yürütülmesi sona erer. Return kullanılmadığında ise fonksiyon varsayılan olarak None değeri döndürür. Bu durum özellikle fonksiyon sonucunun başka bir değişkene atanmak istendiği durumlarda önemlidir.

Fonksiyonlar birden fazla değeri aynı anda da döndürebilir. Bu değerler genellikle bir demet (tuple) yapısı içinde geri gönderilir. Bu özellik, özellikle birden fazla hesaplamanın aynı anda yapılması gereken durumlarda kullanışlıdır.

Değişken Kapsamı, Lambda Fonksiyonlar ve Hata Yönetimi

Fonksiyonlarla birlikte değişken kapsamı kavramı da önem kazanmaktadır. Değişken kapsamı, bir değişkenin programın hangi bölümünde erişilebilir olduğunu ifade eder. Python'da iki temel kapsam bulunmaktadır: yerel kapsam ve genel kapsam.

Yerel kapsam, bir fonksiyon ya da kod bloğu içinde tanımlanan değişkenleri ifade eder. Bu değişkenler yalnızca tanımlandıkları kapsam içinde geçerlidir ve fonksiyon tamamlandığında erişilemez hâle gelirler. Genel kapsam ise fonksiyon dışında tanımlanan değişkenleri kapsar. Global değişkenler programın farklı bölümlerinden erişilebilir. Ancak bir global değişkenin değerini fonksiyon içinde değiştirmek için global anahtar sözcüğünün kullanılması gerekir. Aksi hâlde Python yeni bir yerel

değişken oluşturur.
Aynı isimli yerel ve global değişkenler birbirinden bağımsızdır. Fonksiyon içinde yerel değişken öncelikli olarak kullanılır. Bu durum değişken çakışmalarını önlemek ve programın daha kontrollü çalışmasını sağlamak açısından önemlidir.
Ünitede ayrıca lambda fonksiyonlar ele alınmıştır. Lambda fonksiyonlar, kısa ve tek satırlık işlemler için kullanılan anonim fonksiyonlardır. Genel söz dizimi "lambda parametreler: ifade" şeklindedir. Lambda fonksiyonlarda return ifadesi kullanılmaz; yazılan ifade otomatik olarak döndürülür. Daha karmaşık işlemler için klasik def yapısının kullanılması önerilir.
Son olarak fonksiyonlarda hata yönetimi konusuna değinilmiştir. Program çalışırken oluşabilecek hatalar, istisna mekanizması ile kontrol altına alınabilir. try–except yapısı sayesinde hata oluştuğunda programın tamamen durması engellenir. Özellikle sıfıra bölme gibi hatalarda uygun mesaj verilerek programın güvenli şekilde devam etmesi sağlanabilir. Bu yapı, gerçek uygulamalarda yazılım güvenliği açısından büyük önem taşır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Listeler
ÜNİTE NO 4
YAZAR Dr. ÖZGE ALBAYRAK ÜNAL

GİRİŞ

Programlama sürecinde verilerin nasıl saklandığı, düzenlendiği ve işlendiği temel bir öneme sahiptir. Bir programın sağlıklı biçimde çalışabilmesi, yalnızca doğru komutların yazılmasına değil, aynı zamanda verilerin uygun veri yapıları içerisinde yönetilmesine de bağlıdır. Bu kapsamda liste (list) ve demet (tuple) veri yapıları, birden fazla değerin tek bir yapı altında toplanmasını sağlayan temel araçlar olarak ele alınmaktadır. Bu yapılar sayesinde veriler düzenli, sistematik ve erişilebilir bir biçimde organize edilir. Böylece hem kodun okunabilirliği artar hem de veri üzerinde yapılacak işlemler daha kontrollü bir hâle gelir.

Bu bölümde listelerin ve demetlerin yapısal özellikleri, kullanım biçimleri ve sağladıkları işlevsel kolaylıklar bütüncül bir çerçevede açıklanmıştır.

Liste (List) Veri Yapıları

Listeler, birden fazla veriyi tek bir değişken içerisinde saklamaya olanak tanıyan sıralı veri yapılarıdır. Liste elemanları köşeli parantezler içerisinde tanımlanır ve elemanlar virgül ile ayrılır. Listelerin sıralı yapıda olması, her elemanın belirli bir konuma sahip olduğu anlamına gelir. Bu konumlar indis (indeks) numaraları ile ifade edilir ve indeksler 0'dan başlayarak artar. Ayrıca negatif indeksleme yöntemi ile elemanlara sondan başa doğru erişmek mümkündür.

Listelerin en önemli özelliklerinden biri değiştirilebilir (mutable) olmalarıdır. Bu özellik sayesinde liste oluşturulduktan sonra eleman ekleme, silme ve güncelleme işlemleri doğrudan gerçekleştirilebilir. Bu durum, listeleri dinamik veri işlemleri için oldukça uygun hâle getirir. Listeler aynı zamanda farklı veri türlerini bir arada barındırabilir. Sayısal değerler, metin ifadeleri ya da başka veri yapıları tek bir liste içinde birlikte saklanabilir.

Listeler üzerinde dilimleme (slicing) işlemi uygulanabilir. Bu yöntemle listenin belirli bir aralığına erişilebilir ya da mevcut listenin bir bölümünden yeni bir liste elde edilebilir. Ayrıca "+" operatörü ile iki liste birleştirilebilir, "*" operatörü ile bir liste belirli sayıda çoğaltılabilir. len() fonksiyonu ise listenin kaç elemandan oluştuğunu belirlemek için kullanılır..

Liste Metotları

Listeler üzerinde işlem yapmayı kolaylaştıran çeşitli yerleşik metotlar bulunmaktadır. Bu metotlar, veri üzerinde ekleme, silme, sıralama ve arama işlemlerinin sistematik biçimde gerçekleştirilmesini sağlar.

Python listelerinde append(), extend() ve insert() metotları listeye eleman eklemek amacıyla kullanılır. append() listenin sonuna tek bir eleman eklerken, extend() başka bir yapıdaki birden fazla elemanı listeye dâhil eder. insert() metodu ise belirtilen indekse eleman eklenmesini sağlayarak konuma bağlı düzenleme yapılmasına imkân tanır.

remove() ve pop() metotları eleman silme işlemini gerçekleştirir. remove() belirtilen değeri listeden siler. pop() ise belirtilen indeksteki elemanı siler ve silinen elemanı geri döndürür. Eğer indeks verilmezse son eleman silinir.

Liste sıralama işlemleri sort() metodu ile gerçekleştirilir. reverse() metodu ise mevcut sıralamayı tersine çevirir. count() metodu belirli bir elemanın listede kaç kez tekrarlandığını gösterir. index() metodu ise aranan elemanın kaçıncı indekste bulunduğunu verir. clear() metodu liste içeriğini tamamen boşaltarak listeyi boş hâle getirir. copy() metodu ise listenin bağımsız bir kopyasını oluşturur.

Demet (Tuple) Veri Yapısı

Demetler, listelere benzer biçimde sıralı veri yapılarıdır ve normal parantezler ile tanımlanırlar. Elemanlarına indis numaraları aracılığıyla erişim sağlanabilir. Ancak demetlerin temel farkı değişmez (immutable) olmalarıdır. Oluşturulduktan sonra demet içeriği doğrudan değiştirilemez. Bu özellik, verinin sabit kalmasının gerektiği durumlarda önemli bir avantaj sağlar. Eğer bir demet üzerinde değişiklik yapılması gerekirse, demet önce listeye dönüştürülür, gerekli düzenleme yapılır ve ardından tekrar demet veri tipine çevrilir.

Demetlerin değişmez yapıda olması, onların sözlük veri yapısında anahtar (key) olarak kullanılabilmesini mümkün kılar. Çünkü sözlük anahtarlarının değişmez olması gerekir. Demetlerde temel olarak count() ve index() metotları kullanılmaktadır. Bu metotlar, elemanların tekrar sayısını ve konumunu belirlemek amacıyla kullanılır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Sözlük Veri Tipi
ÜNİTE NO 5
YAZAR Dr.Öğr.Üyesi TUBA IRMAK

•Bu ünitede Python programlama dilinde yer alan sözlük (dictionary) veri tipi ayrıntılı biçimde ele alınmıştır. Sözlükler, verileri anahtar–değer (key–value) çiftleri halinde saklayan ve bir bilgiye anlamlı bir anahtar aracılığıyla erişilmesini sağlayan veri yapılarıdır. Liste yapılarından farklı olarak sözlüklerde erişim indeks numarasına göre değil, anahtar üzerinden gerçekleştirilir. Bu özellik, özellikle ilişkisel verilerin yönetiminde sözlükleri güçlü ve esnek bir yapı haline getirir.
Ünite kapsamında öncelikle sözlük veri tipinin temel tanımı yapılmış ve sözlüklerin süslü parantez kullanılarak nasıl tanımlandığı açıklanmıştır. Anahtarların benzersiz olması gerektiği vurgulanmış; aynı anahtarın birden fazla kez tanımlanması durumunda son değerin geçerli olacağı belirtilmiştir. Ayrıca sözlüklerde anahtar olarak değiştirilemeyen veri tiplerinin kullanılabileceği, liste gibi değiştirilebilir veri tiplerinin anahtar olarak kullanılamayacağı ifade edilmiştir. Buna karşılık değer kısmında farklı veri türlerinin birlikte kullanılabileceği örneklerle gösterilmiştir.
Sözlük veri tipinin kullanım alanlarının daha iyi anlaşılabilmesi amacıyla liste ve sözlük yapıları karşılaştırılmıştır. Liste yapılarında veriye sırasına göre erişilirken, sözlüklerde anlamlı bir anahtar kullanılarak erişim sağlandığı belirtilmiştir. Bu fark, veri organizasyonu ve programlama mantığı açısından önemli bir ayrımı ortaya koymaktadır.
Ünite içerisinde sözlük elemanlarına erişim konusu ayrıntılı biçimde ele alınmış; köşeli parantez kullanımı ve anahtar temelli erişim yöntemi örneklerle açıklanmıştır. Anahtarın sözlükte bulunmaması durumunda oluşabilecek hata türleri incelenmiş ve bu durumun güvenli biçimde yönetilebilmesi için get() metodunun kullanımı gösterilmiştir. Ayrıca setdefault() metodu aracılığıyla sözlükte bulunmayan anahtarlar için varsayılan değer atama işlemi örneklendirilmiştir.
Sözlük metotları işlevlerine göre sınıflandırılarak açıklanmıştır. keys(), values() ve items() metotlarının sözlük içeriğini listeleme amacıyla kullanıldığı; update(), pop() ve clear() metotlarının ise güncelleme ve silme işlemlerini gerçekleştirdiği örnek uygulamalarla gösterilmiştir. Bu metotlar sayesinde sözlük veri yapısının dinamik olarak yönetilebildiği vurgulanmıştır.
Ünitenin ilerleyen bölümünde iç içe sözlük yapıları ele alınmıştır. İç içe sözlüklerde bir anahtarın değerinin başka bir sözlük olabileceği belirtilmiş ve çok katmanlı veri erişiminin hiyerarşik bir yapı izlediği açıklanmıştır. Üst düzey anahtara erişildikten sonra alt sözlük üzerinden ikinci bir anahtar kullanılarak detaylı bilgilere ulaşılabildiği örneklerle gösterilmiştir. Bu yapı, bir varlığa ait birden fazla özelliğin düzenli ve ilişkili biçimde saklanmasına olanak sağlamaktadır.
Son olarak gerçek hayat senaryoları üzerinden sözlük veri tipinin kullanımına yönelik uygulamalar yapılmıştır. Oy sayacı, randevu sistemi, kelime sayacı ve benzeri örnekler aracılığıyla sözlüklerin veri saklama, güncelleme, analiz ve kontrol işlemlerinde nasıl etkin biçimde kullanılabileceği ortaya konmuştur. Böylece öğrenci, sözlük veri tipini yalnızca teorik olarak tanımakla kalmayıp, programlama problemlerinde uygulayabilecek düzeye ulaşmaktadır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Metin Yönetimi
ÜNİTE NO 6
YAZAR Dr. RAZİYE KILIÇ SARIGÜL

METİNLERLE ÇALIŞMA

Python'da metin yönetimini programlamanın temel bir bileşeni olarak ele alır. Günlük uygulamalarda kullanıcıdan alınan girdiler, dosya adları ve uzantılar, rapor çıktıları, web sayfalarından kopyalanan içerikler ve farklı yazılımlar arasında taşınan veriler çoğu zaman str türü üzerinden işlenir. Bu nedenle metin, yalnızca ekranda görülen yazı olarak değil, üzerinde düzenli ve tekrarlanabilir işlemler yapılabilen bir veri yapısı olarak değerlendirilmelidir. Ünite kapsamında metnin Python'da nasıl tanımlandığı ve nasıl temsil edildiği açıklanır. Ardından metni karakter dizisi gibi düşünmenin sağladığı olanaklar üzerinden, metnin belirli kısımlarına erişme, metin içinde arama yapma ve metni parçalara ayırma gibi işlemler ele alınır. Ayrıca metotlar aracılığıyla metni temizleme, hizalama ve daha okunur çıktı üretme gibi pratik dönüşümler üzerinde durulur. Son bölümde ise üretilen çıktının yalnızca ekrana yazdırılmasıyla yetinilmeyip, panoya aktarılıp başka uygulamalara hızlıca taşınabilmesi için pano temelli metin otomasyon yaklaşımı tanıtılır.

METİN SABİTLERİ VE GÖSTERİM BİÇİMLERİ

Python'da metinler tek tırnak veya çift tırnak ile tanımlanabilir ve iki yaklaşım işlevsel olarak aynı amaçla kullanılır. Okunabilirliği artırmak için metnin içeriğine göre uygun tırnak seçimi yapılabilir. Metin içinde hem tek tırnak hem de çift tırnak gibi özel durumlar ortaya çıktığında, yalnızca tırnak seçimi yeterli olmayabilir. Bu tür durumlarda ters bölü ile başlayan kaçış dizileri kullanılır. Kaçış dizileri, karakterin metnin parçası olduğunu açık biçimde belirtmeyi sağlar. Ünite kapsamında sık kullanılan kaçış örnekleri ele alınır. Metin içinde tek tırnak ve çift tırnak yazımı, sekme ve yeni satır üretimi ile ters bölü karakterinin metne eklenmesi gibi durumlar bu çerçevede açıklanır.

Metinlerin sunumunda satır sonları da önemlidir. Tek bir metin içinde yeni satır oluşturmak gerektiğinde \n ile çok satırlı çıktı üretilebilir. Bazı kullanım alanlarında ise ters bölü karakterinin özel anlam kazanması istenmez. Özellikle dosya yolları gibi yapılarda bu durum daha belirgin hale gelir. Bu senaryolarda metnin başına r eklenerek ham metin yaklaşımı kullanılır ve kaçış dizileri işlenmeden korunur. Bunun yanında, uzun metinlerde okunabilirliği artırmak için üç tırnak ile çok satırlı metin yazımı tanıtılır. Üç tırnak kullanıldığında satır sonları metnin parçası olarak korunur ve metin, kod içinde daha düzenli biçimde tutulabilir.

METİNLERDE İNDEKSLEME VE DİLİMLEME

Metinler, karakterlerden oluşan bir yapı olarak ele alınır. Bu bakış açısı, metnin tek bir karakterine erişmeyi ve metnin belirli bir bölümünü almayı mümkün kılar. İndeksleme ile metindeki belirli bir karakter seçilir. Python'da indeksler 0'dan başlar. Bu nedenle ilk karakter s[0] ile ifade edilir. Ayrıca negatif indeksler kullanılarak metnin sonundan geriye doğru erişim sağlanabilir ve bu yöntem, son karakter gibi bölümlere hızlı erişim açısından pratiktir.

Dilimleme ise metnin bir parçasını almak için kullanılır ve başlangıç ile bitiş sınırları üzerinden çalışır. Bitiş sınırının seçime dahil edilmediği yaklaşım özellikle vurgulanır. Başlangıç veya bitiş değerinin boş bırakılması, metnin başını ya da sonunu kapsayacak şekilde yorumlanır. Dilimleme ifadesinin adım bilgisiyle genişletilmesi, metni belirli aralıklarla okumayı ve farklı örüntülerle alt metin elde etmeyi sağlar. Bu bölüm, kullanıcı girdisinden belirli parçaları ayıklama, dosya adı veya uzantı çıkarma gibi yaygın görevlerin mantığını kurmaya yardımcı olur. Ayrıca indeks sınırlarını aşmanın hata üretebileceği, dilimlemede ise sınırların daha esnek yorumlanabildiği belirtilerek temel hata kaynaklarına dikkat çekilir.

METİNLERDE ÜYELİK DENETİMİ

Metin işleme sürecinde sık karşılaşılan gereksinimlerden biri, belirli bir ifadenin bir metin içinde bulunup bulunmadığını kontrol etmektir. Bu amaçla Python'da in ve not in operatörleri kullanılır. in, aranan ifadenin metin içinde yer alıp almadığını test eder ve sonuç olarak mantıksal bir değer üretir. not in ise aynı denetimi ters yönde gerçekleştirir. Bu kontroller, kullanıcıdan alınan bir girdide belirli bir kelimenin geçip geçmediğini doğrulama, bir metnin belirli bir örüntü içerip içermediğini hızlıca sınama gibi işlevlerde temel rol oynar.

Üyelik denetiminde önemli bir nokta, metin karşılaştırmalarında büyük-küçük harf duyarlılığıdır. Bu nedenle arama ve doğrulama adımlarında tutarlı bir değerlendirme için metni belirli bir biçime

dönüştürmek gerekebilir. Ünite, bu tür kontrollerin metin tabanlı programların doğruluğunu ve güvenilirliğini artıran basit fakat etkili adımlar olduğunu gösterir.

METİN METOTLARI İLE DÜZENLEME VE DOĞRULAMA

Bu başlıkta, metinler üzerinde sık kullanılan düzenleme ve kontrol işlemleri ele alınır. Öncelikle, kullanıcı girdilerinde veya kopyalanan metinlerde görülen gereksiz boşlukları gidermek için strip(), lstrip() ve rstrip() metotları kullanılır. Bu metotlar yalnızca boşlukları değil, istenirse belirli bir karakteri de metnin uçlarından kaldırabilir ve temizleme işlemi metnin orta kısmını etkilemez. Metnin biçimsel olarak daha okunur sunulması gerektiğinde ljust(), rjust() ve center() metotlarıyla belirli bir genişlik içinde hizalama yapılır, gerekli durumlarda boşluk yerine farklı bir karakterle doldurma da yapılabilir.

Metin doğrulama amacıyla kullanılan yöntemlerden biri isX biçimindeki metotlardır. Bu metotlar genel olarak "metnin belirli bir özelliği sağlama durumunu" sınar ve sonuç olarak True/False üretir. Bu kapsamda, metnin yalnızca boşluklardan oluşup oluşmadığı isspace() ile; tüm harflerin küçük olup olmadığı islower() ile; tüm harflerin büyük olup olmadığı isupper() ile; başlık biçiminde (kelimelerin ilk harfi büyük) yazılıp yazılmadığı ise istitle() ile denetlenebilir. Bir metnin belirli bir ifadeyle başlayıp başlamadığını ya da bitip bitmediğini kontrol etmek için startswith() ve endswith() metotları kullanılır, bu yaklaşım özellikle uzantı veya kalıp denetimlerinde işlevseldir.
Son olarak metni yapılandırmak için split() ile metin belirli bir ayıraca göre parçalara ayrılır ve sonuç liste olarak elde edilir; parçalar join() ile seçilen ayıraç üzerinden yeniden tek bir metin hâline getirilebilir.

PANOYA KOPYALAMA VE YAPIŞTIRMA İLE METİN OTOMASYON

Metin tabanlı uygulamalarda çıktının yalnızca ekrana yazdırılması her zaman yeterli olmayabilir. Çoğu senaryoda çıktı, e-posta istemcisine, kelime işlemciye veya başka bir programa hızlı biçimde aktarılmak istenir. Bu amaçla pano, program çıktısını dış ortama taşımada pratik bir ara katman oluşturur. Ünite kapsamında bu işlev için pyperclip modülü tanıtılır. Modülün temel kullanımında copy () metni panoya gönderir, paste() ise panodaki güncel içeriği okur ve sonuç olarak bir metin değeri döndürür.

Bu yaklaşım, dosyaya kaydetme gibi ek adımlara ihtiyaç duymadan metin taşımayı hızlandırır. Bununla birlikte pyperclip, Python'un standart kurulumunun bir parçası değildir. Bu nedenle kullanılmadan önce uygun biçimde kurulmuş olmalıdır. Ayrıca pano, programdan bağımsız olarak kullanıcı veya başka uygulamalar tarafından değiştirilebildiği için paste() işleminin her zaman programın en son kopyaladığı veriyi değil, panoda o anda bulunan içeriği getirebileceği dikkate alınmalıdır. Ünite, metin tabanlı küçük araçlar geliştirirken dosyanın başında kısa bir kullanım bilgisini yorum satırı biçiminde vermenin de düzen ve anlaşılabilirlik açısından iyi bir uygulama olduğunu hatırlatır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Nesne Yönelimli Tasarım
ÜNİTE NO 7
YAZAR Dr.Öğr.Üyesi MUSTAFA FURKAN KESKENLER

GİRİŞ

Yazılım geliştirme süreci yalnızca kod yazmaktan ibaret değildir; asıl önemli olan, kod yazılmadan önce sistemin doğru biçimde planlanması ve modellenmesidir. Tasarım, bir yazılımın nasıl çalışacağını, hangi bileşenlerden oluşacağını ve bu bileşenlerin nasıl etkileşim kuracağını belirleyen düşünsel süreçtir. Plansız geliştirilen yazılımlar başlangıçta çalışıyor gibi görünse de zamanla karmaşıklaşır, bakım maliyeti artar ve yeni özellik eklemek zorlaşır. Buna karşılık iyi tasarlanmış sistemler daha düzenli, okunabilir ve sürdürülebilir olur. Özellikle büyük ölçekli projelerde tasarım yapılmadan doğrudan kod yazılması, uzun vadede teknik borç ve yönetilemeyen karmaşıklık oluşturur. Bu nedenle yazılım geliştirme, planlı ve sistematik bir mühendislik süreci olarak ele alınmalıdır.

NESNE YÖNELİMLİ YAKLAŞIMA GENEL BAKIŞ

Nesne yönelimli yaklaşım, yazılımı gerçek dünyadaki varlıkları modelleyerek geliştirmeyi amaçlayan bir tasarım paradigmasıdır. Geleneksel prosedürel programlamada veri ve işlemler ayrı yapılarda tutulurken, nesne yönelimli yaklaşım bu iki unsuru bir araya getirir. Böylece yazılım bileşenleri daha anlamlı ve bütüncül hâle gelir. Bu yaklaşım, karmaşık sistemlerin daha kolay anlaşılmasını sağlar ve yazılımın modüler yapıda geliştirilmesine imkân tanır. Gerçek dünyada varlıkların hem özelliklere hem de davranışlara sahip olması gibi, yazılım nesneleri de hem veri hem de işlev içerir. Bu modelleme biçimi, yazılımın doğal ve sezgisel şekilde tasarlanmasına yardımcı olur.

NESNELER VE SINIFLAR

Nesne yönelimli tasarımın temel yapı taşları sınıf ve nesne kavramlarıdır. Sınıf, benzer nesnelerin ortak özelliklerini ve davranışlarını tanımlayan bir şablondur. Nesne ise bu şablondan oluşturulan somut örnektir ve sistem içinde belirli bir varlığı temsil eder. Aynı sınıftan birçok nesne üretilebilir ve her nesne kendi durum bilgisine sahip olur. Bu yapı, kod tekrarını azaltır ve yazılımın daha düzenli bir şekilde geliştirilmesini sağlar. Sınıf ile nesne arasındaki ilişki, bir mimari proje ile o projeye göre inşa edilen bina arasındaki ilişkiye benzetilebilir. Tasarım sürecinde önce sınıflar belirlenir, ardından bu sınıflardan gerekli nesneler oluşturulur.

NİTELİKLER (ATTRIBUTES) VE DAVRANIŞLAR (METHODS)

Nitelikler (Attributes)

Nitelikler, bir nesnenin sahip olduğu bilgileri ifade eder ve nesnenin durumunu temsil eder. Örneğin bir öğrenci nesnesinin adı, numarası ve not ortalaması onun nitelikleridir. Bu bilgiler nesnenin kimliğini ve özelliklerini belirler.

Davranışlar (Methods)

Davranışlar, nesnenin gerçekleştirebildiği işlemleri ifade eder. Ders seçme, not hesaplama veya bilgi güncelleme gibi işlemler nesnenin davranışlarına örnektir. Davranışlar, nesnenin sistem içindeki işlevini belirler.

Nitelik ve Davranışların Birlikteliği

Nesne yönelimli tasarımın en önemli özelliği, veri ve işlemleri aynı yapı içinde birleştirmesidir. Böylece nesne yalnızca bilgi taşıyan bir yapı olmaktan çıkar, aynı zamanda bu bilgiyi işleyen aktif bir bileşen hâline gelir.

Tasarım Sürecinde Nitelik ve Davranış Belirleme

Bir sınıf tasarlanırken “Bu nesne hangi bilgileri tutmalı?” ve “Bu nesne hangi işlemleri yapmalı?” soruları sorularak nitelik ve davranışlar belirlenir. Doğru seçim, sistemin anlaşılabilirliğini artırır ve gereksiz karmaşıklığı önler.

KAPSÜLLEME VE BİLGİ GİZLEME

Kapsülleme, nesnenin verilerini ve bu veriler üzerinde çalışan işlemleri tek bir yapı içinde toplamasıdır. Bilgi gizleme ise bu yapının dış dünyadan korunmasını sağlar. Nesnenin iç detaylarına doğrudan erişilmez; yalnızca belirlenen metotlar aracılığıyla işlem yapılabilir. Bu yaklaşım veri güvenliğini artırır, hatalı kullanımın önüne geçer ve sistem bileşenleri arasındaki bağımlılığı azaltır. Günlük hayatta bir otomobilin iç mekanizmasını bilmeden kullanılabilmesi gibi, yazılımda da kullanıcı yalnızca gerekli arayüzü görür.

SOYUTLAMA (ABSTRACTİON)

Soyutlama, gereksiz ayrıntıları gizleyip yalnızca önemli özellikleri ön plana çıkarma sürecidir. Karmaşık sistemleri daha sade ve anlaşılır hâle getirir. Geliştirici, sistemin nasıl çalıştığını değil, ne yaptığını modellemeye odaklanır. Bu sayede zihinsel yük azalır ve yazılım daha yönetilebilir olur.

Soyutlama ve kapsülleme birlikte çalışarak hem sade hem de güvenli bir tasarım oluşturur.

SINIFLAR ARASI İLİŞKİLER

Yazılım sistemlerinde sınıflar tek başına çalışmaz; belirli ilişkiler aracılığıyla birlikte hareket eder. Association, Aggregation, Composition ve Inheritance gibi ilişki türleri sınıflar arasındaki bağın niteliğini belirler. Doğru ilişki seçimi, sistemin esnekliğini artırır ve gereksiz bağımlılıkları azaltır. Yanlış ilişkilendirme ise karmaşık ve kırılgan yapılar oluşturabilir. Bu nedenle sınıflar arasındaki bağlantılar dikkatle tasarlanmalıdır.

KALITIM VE ÇOK BİÇİMLİLİK

Kalıtım, bir sınıfın başka bir sınıfın özelliklerini devralmasıdır ve sınıflar arasında hiyerarşik yapı kurulmasını sağlar. Ortak özellikler üst sınıfta toplanır, alt sınıflar bu özellikleri tekrar yazmadan kullanır. Çok biçimlilik ise farklı nesnelerin aynı arayüz üzerinden farklı davranışlar sergilemesine olanak tanır. Bu iki kavram birlikte kullanıldığında yazılım daha esnek ve genişletilebilir hâle gelir. Ancak kalıtım yalnızca gerçek bir "tür" ilişkisi varsa kullanılmalıdır.

İYİ TASARIM İLKELERİ

İyi tasarım, yalnızca çalışan sistemler değil, uzun vadede sürdürülebilir yazılımlar geliştirmeyi hedefler. Tek sorumluluk ilkesi, her sınıfın yalnızca kendi görevine odaklanmasını sağlar. Yüksek uyum ve düşük bağlılık ilkeleri, sınıfların bağımsız ve anlaşılır olmasına yardımcı olur. Gereksiz karmaşıklıktan kaçınmak ve basit çözümleri tercih etmek tasarımın kalitesini artırır. Kodun insanlar tarafından okunacağı unutulmamalı ve anlamlı isimlendirme yapılmalıdır.

TASARIM SÜRECİ

Nesne yönelimli tasarım sistematik bir süreç izlenerek gerçekleştirilir. Önce problem ve gereksinimler analiz edilir, ardından sınıf adayları belirlenir. Daha sonra sınıfların nitelikleri ve davranışları tanımlanır, aralarındaki ilişkiler kurulur ve tasarım sadeleştirilir. Kodlama ise en son aşamada yapılır. Bu yaklaşım hata oranını azaltır ve geliştirme sürecini daha verimli hâle getirir. İyi bir yazılımın temeli, doğru planlanmış bir tasarıma dayanır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Python'da Nesneler
ÜNİTE NO 8
YAZAR Dr.Öğr.Üyesi MUSTAFA FURKAN KESKENLER

GİRİŞ

Önceki ünitede nesne yönelimli tasarımın temel kavramları; problemleri sınıflar, nesneler, nitelikler, davranışlar ve sınıflar arası ilişkiler üzerinden modelleme yaklaşımıyla ele alınmıştı. Amaç, doğrudan kod yazmadan önce problemi doğru anlamak ve mantıklı bir yapı kurmaktı. Çünkü iyi yazılım yalnızca çalışan kod üretmek değildir; doğru tasarlanmış bir model, kodun daha temiz, anlaşılır ve sürdürülebilir olmasını sağlar. Bu ünitede ise tasarımda kurulan kavramsal yapının Python ile somutlaştırılması hedeflenir. Böylece "kavramsal model" ile "çalışan yazılım" arasındaki geçiş adım adım gösterilir. Bu geçiş, birçok öğrenci için kritik bir eşiktir. "Sınıf nedir?" sorusunu yanıtlamak kolaylaşsa da, bunu Python sözdizimiyle doğru şekilde yazmak daha teknik bir süreçtir. Tasarımın soyut yapısı ile kodlamanın somut uygulaması arasında köprü kurulmadığında, tasarım bilgisi pratikte zayıf kalır. Bu ünite, tasarım ile programlama arasındaki bağı güçlendirmeyi amaçlar. Ayrıca Python'un nesne yönelimli yapısı sayesinde öğrenci Python öğrenirken nesne yönelimli düşünmeyi de doğal biçimde pekiştirir. Ünite boyunca nesne kavramını anlamak, sınıf tanımlamak, nesne oluşturmak ve metot eklemek gibi adımlar izlenerek teorik kavramlar çalışan programlara dönüştürülür.

PYTHON'DA NESNE KAVRAMI

Python'da nesne kavramı teorik bir anlatımdan ibaret değildir; dilin temelini oluşturur. Python'da kullandığımız sayılar, metinler, listeler ve fonksiyonlar dâhil pek çok yapı birer nesne olarak ele alınır. Bu nedenle Python öğrenmek, nesne yönelimli yaklaşımı sonradan eklenen bir teknik gibi değil, programlamanın doğal bir parçası olarak öğrenmek anlamına gelir. "Her şey nesnedir" yaklaşımı, nesne yönelimli prensiplerin dilin içine gömülü olduğunu gösterir.

Genel olarak nesne; veriyi (durum/state) ve bu veri üzerinde işlem yapan davranışları (metotları) birlikte barındırır. Prosedürel programlamada veri ile fonksiyonlar çoğu zaman ayrı dururken, nesne yönelimli yaklaşım bu ikisini bir araya getirir. Örneğin bir metin değeri yalnızca bilgi taşımaz; büyütme, parçalama veya uzunluk hesaplama gibi işlemleri kendi üzerinde sunar. Benzer şekilde listeler de eleman ekleme ya da sıralama gibi işlemleri kendi metotlarıyla gerçekleştirir. Bu bütünleşik yapı programcıya daha tutarlı bir çalışma modeli sağlar.

Python'da her nesne bir sınıftan üretilir. Sınıf, nesnenin hangi özelliklere ve davranışlara sahip olacağını belirleyen şablondur; nesne ise bu şablonun somut örneğidir. Ayrıca Python'da değişkenler nesnenin kendisini değil, o nesneye olan referansı tutar. Bu nedenle nesne kavramını doğru anlamak, değişken–nesne ilişkisinin nasıl çalıştığını kavramayı gerektirir. Bu yapı, yerleşik tipler ile kullanıcı tanımlı sınıfların aynı mantıkla çalışmasını sağlayarak öğrenmeyi kolaylaştırır.

DEĞİŞKENLER VE NESNE REFERANSLARI

Python'da değişkenler, birçok dildeki gibi "değeri saklayan kaplar" değildir; bir nesneye referans veren isimlerdir. Bu fark özellikle listeler, sözlükler ve kullanıcı tanımlı nesneler gibi değiştirilebilir yapılarla çalışırken program davranışını doğrudan etkiler. Python'da atama yapıldığında çoğu zaman yeni bir değer kopyalanmaz; değişken mevcut nesneyi işaret edecek şekilde bağlanır. Bu nedenle bir nesneye birden fazla değişken aynı anda referans verebilir.

Bu noktada iki farklı karşılaştırma yaklaşımı önem kazanır: == değer eşitliğini, is ise nesne kimliği eşitliğini ifade eder. İki nesne aynı değerlere sahip olsa bile bellekte farklı varlıklar olarak duruyorsa is sonucu farklı çıkabilir. Bu ayrım, hata ayıklama ve beklenmeyen davranışları açıklama açısından nesne yönelimli programlamada kritik bir rol oynar.

Referans mantığı, değiştirilebilir (mutable) ve değiştirilemez (immutable) türlerde daha belirgindir. Değiştirilemez türlerde yapılan "değişiklik" genellikle yeni nesne üretir; mevcut nesne değişmez. Buna karşılık değiştirilebilir türlerde nesnenin içeriği doğrudan değişir. Bu nedenle aynı nesneyi gösteren iki değişken varsa, birinden yapılan değişiklik diğerinde de görülür. Bu durumu önlemek için bazen referans vermek yerine nesnenin kopyasını oluşturmak gerekir. Her nesnenin bellekte benzersiz bir kimliği bulunur ve bu kimlik id() gibi araçlarla görülebilir. Bu konu, beklenmeyen veri değişimlerini önlemek ve sınıflarla doğru çalışmak için temel bir altyapı sağlar.

TÜR DENETİMİ VE DİNAMİK TİP SİSTEMİ

Python'un üretkenliğini artıran temel özelliklerinden biri dinamik tipli bir dil olmasıdır. Bu sistemde değişkenlerin veri tipi program yazılırken sabitlenmez; program çalıştığında, değişkenin işaret ettiği nesneye göre belirlenir. Bu nedenle değişkenin sabit bir tipi yoktur; tür bilgisi nesnenin kendisine aittir. Aynı değişkenin farklı zamanlarda farklı türde nesneleri göstermesi, esnek ve hızlı geliştirme sağlar.

Tür kontrolü çoğunlukla çalışma sırasında gerçekleştiği için, uyumsuz türlerle bir işlem denenirse hata o anda ortaya çıkar. Örneğin sayı ile metin üzerinde işlem yapılması TypeError ile sonuçlanır. Bu nedenle türleri anlamak ve gerektiğinde kontrol etmek önemlidir. Python'da bir nesnenin türü type() ile görülebilir; belirli bir türe ait olup olmadığı ise isinstance() ile test edilebilir. Özellikle nesne yönelimli kodlarda bu kontroller, daha güvenli çalışma sağlar.
Dinamik tip sistemi daha az sözdizimi, hızlı prototipleme ve yüksek okunabilirlik gibi avantajlar sunar. Ancak hataların çalışma sırasında ortaya çıkması, büyük projelerde tür takibinin zorlaşması gibi riskler de vardır. Bu yüzden iyi tasarım, anlaşılır isimlendirme ve test süreçleri önemlidir.
Tür İpuçları
Dinamik yapıyı bozmadan kodun okunabilirliğini artırmak için Python'da tür ipuçları kullanılabilir. Tür ipuçları, değişkenlerin, fonksiyon parametrelerinin ve dönüş değerlerinin hangi türde olmasının beklendiğini belirtir. Python bu ipuçlarını çalışma anında zorunlu olarak denetlemez; buna rağmen kodu okuyan kişinin fonksiyonları doğru kullanmasını kolaylaştırır.
Tür ipuçları özellikle büyük projelerde faydalıdır: kodun anlaşılması hızlanır, ekip çalışması kolaylaşır ve IDE'ler daha doğru otomatik tamamlama sunar. Ayrıca mypy gibi statik analiz araçları, tür uyumsuzluklarını program çalıştırılmadan önce tespit edebilir. Böylece Python'un esnekliği korunurken daha düzenli bir kod tabanı oluşturulur.

PYTHON'DA SINIF TANIMLAMA

Nesne yönelimli programlamada sınıf, benzer özellik ve davranışlara sahip nesneler için bir şablondur. Python'da gerçek dünyadaki varlıkları modellemek için önce sınıf tanımlanır. Sınıf tanımı class anahtar sözcüğüyle yapılır ve programda yeni bir tür oluşturur.
Sınıflar boş tanımlanabileceği gibi çoğunlukla öznitelik ve metot içerir. Sınıf içindeki metotlar, o sınıftan üretilen tüm nesneler tarafından ortak biçimde kullanılır. Bu metotlarda self parametresi, metotların hangi nesne üzerinde çalışacağını belirler. Ayrıca sınıf isimlendirmesinde okunabilirliği artırmak için PEP 8'e uygun biçimde büyük harfle başlama ve CamelCase yaklaşımı tercih edilir.

NESNE OLUŞTURMA

Sınıf tanımı bir başlangıçtır; asıl kullanılan varlıklar sınıftan üretilen nesnelerdir (instance). Nesne oluşturma işlemi, sınıf adının fonksiyon çağrısı gibi kullanılmasıyla yapılır. Oluşan nesne bellekte yer alır ve bir değişken bu nesneye referans verir.
Aynı sınıftan birden fazla nesne üretilebilir ve her nesne bağımsızdır. Her biri kendi verisini taşır. Bu bağımsızlık, nesne yönelimli yaklaşımın gerçek hayattaki modellemelere uygun olmasının temel nedenlerinden biridir.

Öznitelikler (Attributes)

Öznitelikler, nesnenin durumunu temsil eden verileridir. Bir öğrencinin adı ve numarası, bir arabanın rengi veya bir kitabın yazarı nesne özniteliklerine örnektir. Python'da özniteliklere . notasyonu ile erişilir ve değer atanır. Aynı sınıftan üretilen nesneler aynı türde özniteliklere sahip olabilir; ancak her nesnenin öznitelik değerleri farklı olabilir.
Öznitelikler nesneleri birbirinden ayırır ve metotların işlem yaptığı temel veri alanını oluşturur. Bu nedenle sınıf tasarımında gereksiz öznitelik eklemekten kaçınmak, daha sade ve sürdürülebilir bir yapı sağlar.

Metotlar ve Davranışlar

Nesneler yalnızca veri tutmaz; bu veriler üzerinde işlem yapan davranışlara da sahiptir. Bu davranışları sağlayan fonksiyonlara metot denir. Metotlar sınıf içinde tanımlanır ve nesne üzerinden çağrılır. Böylece nesneler ortak davranışları paylaşır.
Metotların amacı genellikle nesnenin öznitelikleri üzerinde işlem yapmaktır. Örneğin banka hesabı nesnesinde "bakiye" özniteliği varken, "para yatırma/çekme" metotları bu bakiyeyi günceller. Böylece nesne yalnızca veri taşıyan bir yapı değil, sorumluluk sahibi bir varlık hâline gelir.

self Anahtar Sözcüğü

Python'da sınıf içi metotların ilk parametresi geleneksel olarak selftir. self, metodu çağıran nesnenin kendisini temsil eder. Bu sayede metotlar nesneye ait özniteliklere erişebilir. self metot tanımında yer alır; metot çağrılırken yazılmaz çünkü Python bu parametreyi otomatik gönderir.
Bu kavram, metotları normal fonksiyonlardan ayırır. Fonksiyonlar bağımsızdır; metotlar ise nesneye bağlıdır ve nesnenin verisiyle çalışır. Bu mekanizma, her nesnenin kendi durumuyla işlem yapmasını mümkün kılar.

Parametreli Metotlar

Metotların çoğu dışarıdan veri alarak çalışır. Dışarıdan veri alan metotlara parametreli metotlar denir. Parametre kullanımı, nesnelere esneklik kazandırır ve sınıfları daha genel hâle getirir. Parametreli metotlarda selften sonra gelen parametreler, davranışı belirler.
Metotlar birden fazla parametre alabilir, varsayılan değerli parametrelerle isteğe bağlı kullanım sunabilir ve hatta başka nesneleri parametre alarak nesneler arası etkileşim kurabilir. Bu yapı nesne yönelimli tasarımın dinamik ve güçlü yönünü destekler.

NESNE BAŞLATMA

Nesne oluşturulurken çoğu zaman başlangıç değerlerinin belirlenmesi gerekir. Bu amaçla Python'da __ init__ metodu kullanılır. __init__, nesne oluşturulduğu anda otomatik çalışan özel bir metottur ve nesnenin özniteliklerini başlangıçta ayarlayarak tutarlı bir nesne oluşturur.
__init__ kullanılmadığında öznitelikler sonradan tek tek atanır ve bu durum unutmalara, eksik nesnelere ve tutarsızlıklara yol açabilir. __init__ içinde varsayılan değerler tanımlanarak, kullanıcı bazı bilgileri vermediğinde sistemin güvenli başlangıç değerleriyle nesne üretmesi sağlanabilir.

MODÜLLER VE PAKETLER

Kod büyüdükçe her şeyi tek dosyada tutmak okunabilirliği düşürür ve bakımını zorlaştırır. Bu nedenle Python'da kodu düzenlemek için modül ve paket yapıları kullanılır. Modül, en basit hâliyle bir Python dosyasıdır (.py) ve ilgili kodlar aynı dosyada toplanır. Paket ise modüllerin klasörler altında gruplanmış hâlidir ve büyük projelerde daha güçlü bir organizasyon sağlar.
Küçük projelerde tek dosya yeterli olabilir; orta ölçekli projelerde modüller düzen sağlar; büyük projelerde ise paket + modül yapısı gerekir. Bu yapı ekip çalışmasını kolaylaştırır, tekrar kullanım sağlar, hata ayıklamayı hızlandırır ve projenin profesyonel ölçekte yönetilmesini destekler.

Nesne yönelimli programlama, yazılım sistemlerini gerçek dünyadaki varlıkları modelleyerek daha anlaşılır, düzenli ve sürdürülebilir hâle getirmeyi amaçlayan bir yaklaşımdır. Bu yaklaşımın temelinde, veri ve davranışı bir arada tutan nesne kavramı yer alır. Nesne, belirli özelliklere ve bu özellikler üzerinde işlem yapabilen davranışlara sahip yazılım bileşenidir. Gerçek dünyadaki varlıkların yazılım ortamındaki temsili olarak düşünülebilir.
Nesnelerin türlerini ve ortak özelliklerini tanımlamak için sınıf kavramı kullanılır. Sınıf, benzer özellik ve davranışlara sahip nesnelerin ortak şablonudur. Nesneler, sınıflardan üretilir ve her nesne kendi durum bilgisine sahip olabilir. Bu yapı, yazılımın düzenli ve modüler biçimde tasarlanmasını sağlar. Aynı sınıfa ait birden fazla nesne bulunabilir ve her biri farklı değerler taşıyabilir. Bu yönüyle sınıflar, sistemin mimari temelini oluşturur.
Nesne yönelimli programlamada nesneler genellikle tek başına değil, diğer nesnelerle kurdukları ilişkiler aracılığıyla anlam kazanır. Bu nedenle nesneler arası ilişkiler, yazılım tasarımının merkezinde yer alır. İlişkilendirme (association), iki sınıfa ait nesnelerin birbirleriyle bağlantılı olduğunu ifade eder. Bu bağlantı, bir nesnenin başka bir nesneyi kullanması, onunla bilgi alışverişinde bulunması veya birlikte bir süreci gerçekleştirmesi şeklinde olabilir. UML (Unified Modeling Language) diyagramları, sınıflar ve aralarındaki ilişkileri görsel olarak ifade etmek için kullanılan standart bir modelleme aracıdır. UML'de ilişkiler çizgilerle, çokluk (multiplicity) ise sayısal ifadelerle gösterilir. Çokluk kavramı, bir nesnenin başka bir nesneyle kaç adet ilişki kurabileceğini tanımlar.

Bütün–parça ilişkileri bağlamında iki önemli kavram olan bileşim (composition) ve birleştirme (aggregation) ayrıntılı biçimde incelenmiştir. Bileşim, parça nesnelerin bütüne güçlü biçimde bağlı olduğu bir ilişki türüdür. Bu durumda parça nesneler, bütün nesne olmadan varlıklarını sürdüremez. Bütün ortadan kalktığında parça da yok olur. Bu, güçlü sahiplik ilişkisini ifade eder. Birleştirme ise daha gevşek bir bütün–parça ilişkisidir. Parça nesneler bütünden bağımsız olarak oluşturulabilir ve varlıklarını sürdürebilir. Bu ayrım, nesnelerin yaşam döngüsü üzerinden değerlendirilir ve tasarım kararlarında önemli rol oynar. Doğru ilişki türünün seçilmesi, sistemin sorumluluk dağılımını netleştirir ve bakım sürecini kolaylaştırır.
Kalıtım (inheritance), nesne yönelimli programlamanın temel soyutlama mekanizmalarından biridir. Kalıtım, sınıflar arasında "is-a" ilişkisi kurarak özellik ve davranışların yeniden kullanılmasını sağlar. Üst sınıf (superclass), daha genel bir kavramı temsil ederken; alt sınıf (subclass), bu kavramın daha özelleşmiş biçimini ifade eder. Bu yapı sayesinde ortak özellikler üst sınıfta toplanır, alt sınıflar yalnızca kendilerine özgü davranışları ekler. Kalıtımın temel amacı; kod tekrarını azaltmak, ortak davranışları merkezileştirmek ve genişletilebilir bir sistem mimarisi oluşturmaktır.
Kalıtım ile doğrudan ilişkili olan soyutlama (abstraction), karmaşık sistemlerin yönetilebilir hâle getirilmesini sağlar. Soyutlama, gereksiz ayrıntıların gizlenmesi ve yalnızca gerekli özelliklerin öne çıkarılması sürecidir. Soyut sınıflar (abstract classes), alt sınıflar için bir davranış çerçevesi oluşturur. Üst sınıf, belirli metotların varlığını zorunlu kılar ancak bu metotların ayrıntılı uygulamasını içermez. Böylece alt sınıflar belirlenen kurallara uyarak davranışı somutlaştırır. Bu yaklaşım, sistemin tutarlılığını korur ve genişletilebilirliğini artırır.
Metot ezme (method overriding), kalıtımın sağladığı önemli mekanizmalardan biridir. Alt sınıf, üst sınıfta tanımlanmış bir metodu yeniden tanımlayarak davranışı özelleştirebilir. Bu sayede sistemin dış arayüzü sabit kalırken iç davranış değiştirilebilir. Bu özellik, yazılım sistemlerinde esneklik ve davranış çeşitliliği sağlar.
Polimorfizm (polymorphism), aynı arayüzün farklı nesneler tarafından farklı biçimlerde uygulanabilmesini ifade eder. Aynı metod çağrısı, farklı nesnelerde farklı sonuçlar üretebilir. Bu yaklaşım sayesinde sistem, nesnenin türünü bilmeden ortak bir davranışı çağırabilir. Polimorfizm; esneklik, modülerlik ve genişletilebilirlik açısından büyük avantaj sağlar. Yeni türler, mevcut yapıyı bozmadan sisteme eklenebilir ve kod tekrarının önüne geçilir.
Python diline özgü önemli bir tasarım yaklaşımı olan duck typing, tür hiyerarşisinden ziyade davranışın varlığına odaklanır. Bir nesne gerekli davranışı sunuyorsa, hangi sınıfa ait olduğuna bakılmaksızın kullanılabilir. Bu yaklaşım, Python'un dinamik tip sisteminin doğal bir sonucudur ve daha gevşek bağlı sistemler oluşturulmasına imkân tanır. Tür kontrolü yerine davranış kontrolü esas alınır. Bu da

yazılımın değişime daha açık ve esnek olmasını sağlar.
Çoklu kalıtım (multiple inheritance), bir sınıfın birden fazla üst sınıftan miras alabilmesini ifade eder. Bu yapı, farklı davranış kümelerinin tek bir sınıf altında toplanmasını sağlar. Ancak beraberinde hiyerarşik karmaşıklık ve metot çakışması gibi sorunlar getirebilir. Python'da bu durum Metot Çözümleme Sırası (MRO) mekanizması ile yönetilir. MRO, hangi metodun hangi sırayla çağrılacağını belirleyen kurallar bütünüdür. Tasarım ilkesi olarak, mümkün olan durumlarda kalıtım yerine bileşimin tercih edilmesi önerilir.
Sonuç olarak bu ünitede nesne yönelimli programlamanın kuramsal temelleri; nesne, sınıf, ilişkilendirme, çokluk, bileşim, birleştirme, kalıtım, soyutlama, soyut sınıf, metot ezme, polimorfizm, duck typing ve çoklu kalıtım kavramları üzerinden sistematik biçimde ele alınmıştır. Bu kavramların doğru anlaşılması, yazılım sistemlerinin daha düzenli, modüler, esnek ve sürdürülebilir biçimde tasarlanmasını sağlar. Nesne yönelimli yaklaşım, karmaşık sistemleri daha yönetilebilir hâle getirerek hem geliştirme sürecini hem de bakım sürecini kolaylaştırır. Bu nedenle modern yazılım mühendisliğinde nesne yönelimli tasarım, temel ve vazgeçilmez bir paradigma olarak kabul edilmektedir.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI İstisna Yönetimi
ÜNİTE NO 10
YAZAR Arş.Gör.Dr Rabiye KILIÇ

Bu bölümde yazılım geliştirme sürecinde karşılaşılan hata kavramı ve Python programlama dilindeki istisna (exception) mekanizması kapsamlı biçimde ele alınmıştır. Programların her zaman beklenen şekilde çalışmayabileceği gerçeğinden hareketle, kullanıcı hataları, eksik veri girişleri, sistem kaynaklı problemler ve tasarım eksikliklerinin yazılımın normal akışını bozabileceği vurgulanmıştır. Bu tür durumların kontrolsüz biçimde ortaya çıkmasının programın ani şekilde sonlanmasına ve veri kaybına yol açabileceği belirtilmiş; bu nedenle hataların sistematik ve kontrollü biçimde yönetilmesinin yazılım güvenilirliği açısından kritik olduğu ifade edilmiştir.
Bölümde öncelikle hata (error) ve istisna (exception) kavramları açıklanmış, bu iki terim arasındaki teknik fark ortaya konulmuştur. Hatanın daha genel bir problem durumunu ifade ettiği; istisnanın ise bu problemin programlama dili tarafından tanımlanmış ve yönetilebilir hâle getirilmiş biçimi olduğu belirtilmiştir. "Her istisna bir hata durumudur; ancak her hata bir istisna değildir" ilkesi temel ayrım noktası olarak vurgulanmıştır. Özellikle sözdizimsel ve mantıksal hataların istisna mekanizması ile her zaman yakalanamayacağı, buna karşılık çalışma zamanı hatalarının Python'da istisna olarak üretildiği açıklanmıştır.
Hata türleri; sözdizimsel hatalar, mantıksal hatalar ve çalışma zamanı hataları olmak üzere üç ana başlık altında incelenmiştir. Sözdizimsel hataların program çalıştırılmadan önce tespit edildiği, mantıksal hataların ise program çalışsa bile yanlış sonuç üretmesine neden olduğu belirtilmiştir. Çalışma zamanı hatalarının ise programın gerçek kullanım sürecinde ortaya çıktığı ve bu nedenle istisna yönetimi açısından en kritik hata türü olduğu ifade edilmiştir.
Python'da istisna yapısının temel bileşenleri olan try–except, else ve finally blokları ayrıntılı biçimde açıklanmıştır. try bloğunun hata oluşma ihtimali bulunan kodları içerdiği, except bloğunun oluşan istisnayı yakaladığı, else bloğunun hata oluşmadığında çalıştığı ve finally bloğunun ise hata oluşsa da oluşmasa da her durumda yürütüldüğü belirtilmiştir. Bu yapı sayesinde programın ani şekilde sonlanmasının önüne geçildiği ve hata yönetiminin kontrollü biçimde gerçekleştirildiği vurgulanmıştır.
Bölümde ayrıca istisnaların yükseltilmesi (raise) kavramı ele alınmıştır. Python'un yalnızca oluşan hatalara tepki vermekle kalmayıp, geliştiricinin belirli mantıksal koşullar altında bilinçli olarak istisna üretmesine de imkân tanıdığı belirtilmiştir. Bu yaklaşımın özellikle iş kurallarının korunması, veri doğrulama süreçleri ve sistem bütünlüğünün sağlanması açısından önemli olduğu ifade edilmiştir.
Özel istisna sınıfları konusu kapsamında, uygulamaya özgü hata durumlarının daha anlamlı ve düzenli biçimde ifade edilebilmesi için yeni istisna sınıflarının tanımlanabileceği açıklanmıştır. Bu sınıfların genellikle Exception sınıfından türetildiği ve hata yönetimini daha okunabilir, modüler ve sürdürülebilir hâle getirdiği belirtilmiştir. Ayrıca istisnaların belirli bir kalıtım yapısı içinde organize edildiği ve Python'daki istisna hiyerarşisinin hata yönetimine esneklik kazandırdığı vurgulanmıştır.
Son olarak iyi istisna kullanım ilkeleri üzerinde durulmuştur. İstisnaların yalnızca olağan dışı durumlar için kullanılması, mümkün olduğunca spesifik biçimde yakalanması, doğru seviyede ele alınması ve anlamlı hata mesajları üretilmesi gerektiği ifade edilmiştir. Bilinçli ve disiplinli bir istisna yönetiminin yazılımın güvenilirliğini, okunabilirliğini ve bakım kolaylığını artırdığı sonucuna ulaşılmıştır.
Genel olarak bu bölüm, Python'da istisna mekanizmasının yalnızca teknik bir hata yakalama aracı değil, aynı zamanda yazılım tasarımının önemli bir parçası olduğunu ortaya koymaktadır. Doğru yapılandırılmış bir istisna yönetimi, sağlam, esnek ve profesyonel yazılım sistemlerinin geliştirilmesinde temel bir rol oynamaktadır.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Nesne Yönelimli Programlama ne zaman kullanılır?
ÜNİTE NO 11
YAZAR Arş.Gör.Dr Rabiye KILIÇ

Bu bölümde nesne yönelimli programlama (Object-Oriented Programming – OOP) yaklaşımının yazılım geliştirme sürecindeki rolü ve hangi durumlarda tercih edilmesi gerektiği ele alınmıştır. Yazılım sistemlerinin giderek daha karmaşık hâle gelmesi, programların yalnızca işlemler dizisi şeklinde tasarlanmasının yetersiz kalmasına neden olmuştur. Bu nedenle modern yazılım geliştirme süreçlerinde nesne yönelimli programlama yaklaşımı yaygın olarak kullanılmaktadır. Bu yaklaşım, programları birbirleriyle etkileşim hâlinde olan nesnelerden oluşan bir yapı olarak ele alarak yazılım sistemlerinin daha düzenli, modüler ve sürdürülebilir bir biçimde geliştirilmesine olanak sağlamaktadır.

Bölümün ilk kısmında nesneleri gerçek dünyadaki varlıklar gibi düşünmenin önemi üzerinde durulmuştur. Nesne yönelimli programlama yaklaşımında yazılım sistemleri, gerçek dünyadaki kavramların yazılım ortamında modellenmesiyle oluşturulur. Bu modelleme sürecinde her nesne belirli bir durumu temsil eden veriler ile bu veriler üzerinde işlem yapan davranışları birlikte içerir. Bu yaklaşım sayesinde yazılım sistemleri daha anlaşılır hâle gelir ve programın farklı bölümleri arasında daha düzenli bir görev dağılımı sağlanır. Ayrıca bu yapı, modülerlik ilkesini destekleyerek yazılımın farklı bileşenlerinin bağımsız olarak geliştirilebilmesine ve gerektiğinde kolaylıkla değiştirilebilmesine olanak tanır.

Bölümde ayrıca sınıf verilerine davranış eklemek için kullanılan property kavramı ayrıntılı olarak incelenmiştir. Property mekanizması, bir sınıf içerisindeki değerlere değişken gibi erişilebilmesini sağlarken aynı zamanda arka planda belirli hesaplama veya kontrol işlemlerinin yapılmasına olanak tanır. Bu sayede sınıf içindeki veri erişimi daha kontrollü hâle getirilebilir ve veri güvenliği artırılabilir. Özellikle hesaplanan değerlerin tanımlanması veya belirli kuralların uygulanması gerektiği durumlarda property yapısı önemli bir kolaylık sağlamaktadır.

Property mekanizmasının daha düzenli bir şekilde kullanılabilmesi için Python programlama dilinde kullanılan decorator yapısı da bölümde ele alınmıştır. Decorator, bir fonksiyonun veya metodun davranışını değiştiren ya da genişleten özel bir yapıdır. Property ile birlikte kullanılan decorator yapıları sayesinde veri erişimi üzerinde daha kapsamlı kontrol sağlanabilir. Getter, setter ve deleter gibi yapılar aracılığıyla sınıf içindeki değerlerin okunması, değiştirilmesi ve silinmesi işlemleri kontrollü bir şekilde gerçekleştirilebilir. Bu durum yazılım sistemlerinin daha güvenilir ve esnek bir yapıya sahip olmasına katkı sağlar.

Bölümde ayrıca nesne yönelimli tasarımda önemli bir kavram olan manager nesneleri ele alınmıştır. Manager nesneleri, sistemde bulunan diğer nesneleri yönetmek ve aralarındaki ilişkileri düzenlemek amacıyla kullanılan yapılardır. Büyük yazılım projelerinde tüm işlemlerin tek bir sınıf içerisinde gerçekleştirilmesi yerine görevlerin farklı nesnelere dağıtılması ve bu nesnelerin bir yönetici yapı tarafından kontrol edilmesi yazılım mimarisinin daha düzenli olmasını sağlar. Manager nesneleri sayesinde sistemdeki farklı bileşenler arasında koordinasyon sağlanabilir ve programın bakım süreçleri daha kolay hâle getirilebilir.

Bölümün önemli konularından biri de yazılım tasarımında kullanılan DRY (Don't Repeat Yourself) prensibidir. Bu prensip, yazılım sistemlerinde aynı kod parçalarının tekrar edilmemesi gerektiğini ifade eder. Kod tekrarının fazla olduğu sistemlerde bakım ve güncelleme işlemleri daha zor hâle gelir ve hata yapma olasılığı artar. Bu nedenle yazılım tasarımında tekrar eden kod parçalarının tek bir yerde tanımlanması ve gerektiğinde yeniden kullanılabilir hâle getirilmesi önemlidir. Nesne yönelimli programlama yaklaşımında kullanılan kalıtım ve bileşim gibi tasarım teknikleri, kod tekrarını azaltmak ve daha düzenli bir yazılım mimarisi oluşturmak için etkili yöntemler sunmaktadır.

Son olarak bölümde sınıf hiyerarşisi ve veri doğrulama kavramları ele alınmıştır. Sınıf hiyerarşisi, yazılım sistemlerinde sınıflar arasında genelden özele doğru kurulan ilişkileri ifade eder. Bu yapı sayesinde ortak özellikler üst sınıflarda tanımlanırken alt sınıflar daha özel özellikler ve davranışlar ekleyebilir. Böylece yazılım sistemlerinde hem kod tekrarının azaltılması hem de daha düzenli bir yapı oluşturulması mümkün olur. Veri doğrulama ise yazılım sistemlerinde kullanılan verilerin doğruluğunu kontrol etmek amacıyla kullanılan önemli bir mekanizmadır. Hatalı veri girişlerinin kontrol edilmesi, programın güvenilirliğini artırır ve sistemin beklenmeyen hatalar üretmesini engeller.

Sonuç olarak bu bölümde ele alınan kavramlar, nesne yönelimli programlama yaklaşımının yalnızca bir programlama tekniği olmadığını, aynı zamanda yazılım tasarımını daha sistematik ve

sürdürülebilir hâle getiren önemli bir yöntem olduğunu göstermektedir. Nesnelerin doğru şekilde modellenmesi, veri erişiminin kontrol altına alınması, kod tekrarının azaltılması ve sınıflar arasında düzenli ilişkiler kurulması yazılım sistemlerinin daha kaliteli ve güvenilir bir biçimde geliştirilmesine katkı sağlamaktadır. Bu nedenle nesne yönelimli programlama yaklaşımı günümüzde birçok modern yazılım sisteminin temel tasarım yaklaşımı olarak kullanılmaktadır.

DOSYALAR VE DOSYA YOLLARI

Bu ünitede dosya işlemleri, dosya adının tek başına yeterli olmadığı ve çoğu zaman dosya yolunun doğru biçimde tanımlanması gerektiği fikri üzerinden açıklanır. Dosyalar, klasörler içinde hiyerarşik biçimde tutulur ve her dosya, dosya adı ile birlikte onu konumlandıran bir yol bilgisine sahiptir. Dosya adının son noktasından sonra gelen uzantı, dosya türüne ilişkin bir ipucu verir. Kök klasör gösterimi işletim sistemine göre değişir. Windows ortamında kök çoğunlukla sürücü harfiyle ifade edilirken macOS ve Linux ortamlarında kök dizin eğik çizgi ile gösterilir. Bir diğer temel fark yol ayırıcı karakteridir. Windows ters bölü kullanırken macOS ve Linux düz bölü kullanır. Bu farklılık, yolu elle birleştiren kodlarda uyumluluk sorunları doğurabilir. Bu nedenle dosya ve klasör adlarını birleştirirken işletim sistemine uygun ayırıcıyı otomatik seçen bir yaklaşım tercih edilir. Bu amaçla os.path.join() ile yol parçaları güvenli biçimde birleştirilebilir.

ÇALIŞMA DİZİNİ VE YOLLARIN YORUMLANMASI

Dosya işlemlerinde temel kavramlardan biri çalışma dizinidir. Python, göreli yolları yorumlarken başlangıç noktası olarak çalışma dizinini esas alır. Bu nedenle aynı göreli yol ifadesi, çalışma dizini değiştirildiğinde farklı bir dosya ya da klasörü işaret edebilir. Çalışma dizinini görmek için os.getcwd (), çalışma dizinini değiştirmek için os.chdir() kullanılır. Var olmayan bir dizine geçilmeye çalışıldığında hata oluşması, işlem öncesinde yol denetiminin önemini gösterir. Dosya yolu belirtmede iki temel yaklaşım bulunur. Mutlak yol kök dizinden başlayarak konumu baştan sona tanımlar. Göreli yol ise çalışma dizinine göre anlam kazanır. Bu bağlamda nokta(.) mevcut dizini, iki nokta (..) ise üst klasörü temsil eder. Göreli yollarda başlangıçtaki nokta ve ayırıcı ifadesi çoğu zaman isteğe bağlıdır. Yeni bir klasör yapısı gerektiğinde os.makedirs() ile yol boyunca eksik ara klasörler de oluşturulabilir ve böylece hedef yolun varlığı sağlanır.

os.path MODÜLÜ

os.path altında yer alan işlevler, dosya yollarını daha güvenli ve anlaşılır biçimde işlemeyi sağlar. Dosya yolları tek bir metin gibi görünse de çoğu zaman iki ana bileşenden oluşur. Bunlar dosyanın bulunduğu dizin yolu ve dosya adı ile uzantının birlikte yer aldığı temel addır. Bu ayrımı yapabilmek, örneğin yalnızca klasör yoluyla işlem yürütme veya dosya adını değiştirme gibi uygulamalarda gereklidir. Bu amaçla dosya adını almak için os.path.basename(), klasör yolunu almak için os.path.dirname() kullanılabilir. Her iki bilgiyi birlikte elde etmek gerektiğinde os.path.split() tercih edilebilir. Yol içindeki klasörleri tek tek incelemek istendiğinde, işletim sistemine bağlı yol ayırıcı karakteri os.path.sep ile belirlenir ve yolun bu ayırıcıya göre parçalanması daha güvenli bir yaklaşım sunar.

os.path, yolun türünü ve biçimini değerlendirmeye yönelik denetimler de sağlar. Bir yolun mutlak yol olup olmadığı os.path.isabs() ile kontrol edilir. Göreli bir yol, os.path.abspath() ile çalışma dizinine göre mutlak yola dönüştürülerek belirsizlik azaltılabilir. Ayrıca bir hedefe belirli bir başlangıç konumundan hangi göreli yolla gidileceği os.path.relpath() ile elde edilir ve bu yaklaşım klasörler arası geçiş mantığını açıklığa kavuşturur.

Dosya işlemlerinde sık karşılaşılan sorunlardan biri, erişilmek istenen yolun sistemde bulunmamasıdır. Bir diğer sorun ise verilen yolun dosyayı mı yoksa klasörü mü gösterdiğinin belirsiz olmasıdır. Bu nedenle işlem öncesinde yolun varlığını ve türünü doğrulamak önemlidir. Yolun var olup olmadığını sınamak için os.path.exists() kullanılır. Yolun bir dosyayı mı yoksa bir dizini mi gösterdiğini ayırt etmek için os.path.isfile() ve os.path.isdir() ile tür denetimi yapılır. Bu denetimler True veya False döndürdüğü için karar yapılarıyla birlikte güvenli bir ön kontrol oluşturur.

os.path ve os modülündeki ilgili işlevler, dosyayı açmadan da bazı bilgileri elde etmeyi mümkün kılar. Bir klasörün içeriğini görmek için os.listdir() ile ad listesi alınabilir. Dosya boyutunu öğrenmek için os.path.getsize() çağrısı yapılır. Çok sayıda dosyanın toplam boyutunu hesaplamak gerektiğinde ise klasör içeriğindeki öğelerin tam yolu oluşturularak boyutların toplanması yoluna gidilir. Bu tür hesaplamalarda alt klasörlerin bulunabileceği dikkate alınmalı ve gerekirse yolun dosya mı klasör mü olduğu ayrıca denetlenmelidir.

DOSYA OKUMA VE YAZMA SÜRECİ

Dosya okuma ve yazma süreci, temel olarak düz metin dosyaları üzerinden açıklanır. Düz metin dosyaları yalnızca karakter içerir ve yazı tipi, renk ya da biçimlendirme gibi ek bilgiler taşımaz. Buna karşılık PDF, görsel dosyalar, yürütülebilir dosyalar ve ofis belgeleri gibi ikili dosyalar farklı bir

yapıya sahiptir. İkili dosyalar, doğrudan okunabilir karakterlerden oluşmaz ve içerikleri dosya formatının kurallarına göre düzenlenmiş veriler içerir. Bu nedenle ikili dosyalar, düz metin dosyalarıyla kullanılan yöntemlerle doğrudan aynı biçimde işlenmeye uygun değildir.
Python'da düz metin dosyalarıyla çalışmanın temel akışı dosyayı açma, okuma ya da yazma işlemini gerçekleştirme ve işlem tamamlandığında dosyayı kapatma adımlarından oluşur. open() fonksiyonuna dosyanın yolu verildiğinde Python, dosyaya erişimi sağlayan bir dosya nesnesi döndürür. Yol mutlak ya da göreli olabilir. Varsayılan davranış dosyayı okuma modunda açmaktır ve gerekirse okuma modu 'r' açık biçimde belirtilebilir. Dosyanın tüm içeriğini tek seferde almak için read() kullanılabilir. Dosyayı satır satır liste biçiminde elde etmek için readlines() tercih edilir. Satır temelli okumada, satır sonlarının çoğu zaman yeni satır karakteriyle birlikte geldiği unutulmamalıdır.
Dosyaya yazmak için dosyanın uygun modda açılması gerekir. Yazma modu çoğu durumda dosyayı yeniden oluşturur ya da varsa içeriği silerek baştan yazar. Ekleme modu ise var olan içeriğin sonuna yeni metin ekler. Yazma işlemi write() ile yapılır ve yeni satırın otomatik eklenmediği durumlarda gerekli satır sonu karakteri metne dâhil edilir. Okuma veya yazma tamamlandıktan sonra close() ile dosya kapatılarak program ile dosya arasındaki bağlantı sonlandırılır. Dosyanın her durumda kapatılmasını güvence altına almak için with open(...) as ... yapısı kullanılabilir. Bu yapı, blok tamamlandığında dosyayı otomatik kapatarak kaynak yönetimini kolaylaştırır ve kapatma adımının unutulmasıyla oluşabilecek sorunları azaltır.

DEĞİŞKENLERİ DOSYADA KALICI OLARAK SAKLAMA

Bu bölümde, verinin yalnızca düz metin olarak saklanmasıyla yetinilmeyip program içinde yeniden kullanılabilecek biçimde kalıcı olarak depolanması ele alınır. Düz metin dosyaları kullanıcı tarafından okunabilir içerik üretmek için uygundur. Ancak program içinde üretilen liste veya sözlük gibi veri yapılarının yalnızca metin olarak yazılması her zaman yeterli olmayabilir. Bu nedenle iki temel yaklaşım öne çıkar.
İlk yaklaşım, Python nesnelerini dosya sisteminde ikili biçimde saklayan kalıcı sözlük yapısıdır ve bu amaçla shelve modülü kullanılır. Oluşturulan depo, kullanım açısından bir sözlük gibi davranır. Veriler anahtarlar üzerinden kaydedilir ve daha sonra aynı anahtarlarla geri çağrılabilir. Depodaki anahtarlar ve değerler keys() ve values() aracılığıyla elde edilebilir. İşletim sistemine göre bir veya birden fazla yardımcı dosyanın oluşması olağan kabul edilir. İşlem tamamlandığında deponun kapatılması, kaydın tamamlanması ve kaynak yönetimi açısından gereklidir.
İkinci yaklaşım, veriyi okunabilir biçimde arşivlemek amacıyla liste veya sözlük gibi yapıların düzenli bir metne dönüştürülmesidir. Bu amaçla pprint.pformat() kullanılarak veri yapısının Python sözdizimine uygun, daha okunur bir metin temsili üretilebilir. Üretilen metin bir Python dosyasına değişken tanımı biçiminde yazıldığında, dosya daha sonra bir modül gibi içe aktarılarak veriye program içinden erişilebilir. Böylece saklanan veri hem okunabilir bir biçimde korunur hem de gerektiğinde doğrudan program tarafından kullanılabilir.

DERS ADI Nesne Tabanlı Programlama II
ÜNİTE ADI Excel Dosya Yönetimi
ÜNİTE NO 13
YAZAR Dr. ÖZGE ALBAYRAK ÜNAL

GİRİŞ

Elektronik tablolar, veriyle çalışan birçok alanda yaygın olarak kullanılır. Finans kayıtları, stok takibi, not listeleri ve istatistiksel raporlar gibi işlemler bu araçlarla yürütülebilir. Ancak veri arttıkça kayıt bulma, toplu değişiklik yapma, sınıflandırma ve özetleme gibi işler zaman alır ve hata riskini yükseltir. Bu yüzden büyük dosyalarda manuel çalışma verimsizleşir.

Excel, Windows'ta sık kullanılan bir elektronik tablo uygulamasıdır ve .xlsx formatını destekleyen farklı platformlarda çalışan alternatifleri de vardır. Tekrar eden işlemleri hızlandırmanın ve hataları azaltmanın etkili yolu ise programlamadır. Python, otomasyon için uygun yapısı ve kütüphane desteği sayesinde bu tür işleri kolaylaştırır.

Excel dosyalarıyla Python üzerinden çalışmak için openpyxl kütüphanesi kullanılabilir. openpyxl, .xlsx dosyalarını açma, okuma, düzenleme ve kaydetme imkânı vererek hücrelere erişmeyi, satır-sütunlarda işlem yapmayı ve verileri otomatik olarak güncellemeyi sağlar. Böylece veri aktarma, filtreleme ve çok sayıda dosyada aynı işlemi uygulama gibi tekrar eden görevler otomatikleştirilebilir.

Çalışma Ortamının Hazırlanması

Python ile Excel dosyaları üzerinde işlem yapabilmek için önce teknik hazırlıkların tamamlanması gerekir. Bu kapsamda openpyxl kütüphanesinin kurulumu ve kurulumun doğrulanması temel bir adımdır. Python, openpyxl kütüphanesini varsayılan olarak içermediği için .xlsx dosyalarıyla çalışmaya başlamadan önce ayrıca kurulması gerekir.

Kurulumun doğru tamamlandığını doğrulamanın en temel yolu, kütüphaneyi Python ortamında içe aktarmaktır. Kurulum başarılıysa içe aktarma sırasında hata alınmaz. Buna karşılık openpyxl içe aktarılmadan kullanılmaya çalışılırsa NameError türünde hatalar görülebilir.

Hazırlık sürecinde çalışma dizini ve dosya yolu kullanımı da önemlidir. Excel dosyasının bulunduğu konum ile Python'un geçerli çalışma dizini arasındaki ilişki doğru kurulmazsa dosyaya erişimde sorun yaşanabilir. Bu nedenle uygulama örneklerinin tutarlı biçimde çalışabilmesi için dosyaların doğru dizinde bulunması ve dosya yolunun doğru kullanılması gerekir.

Excel Belgelerini Okuma

Uygulama örneklerinde ornek.xlsx adlı bir Excel dosyası kullanılır. Örnek dosyada, yeni çalışma kitaplarında elektronik tablo yazılımlarının varsayılan olarak oluşturduğu Sheet1, Sheet2 ve Sheet3 adlı sayfalar yer alır. Varsayılan sayfa sayısının işletim sistemine ve kullanılan programa göre değişebileceği belirtilir.

Dosyayla çalışmak için openpyxl içe aktarıldıktan sonra load_workbook() fonksiyonu ile çalışma kitabı yüklenir. Bu fonksiyon dosya adını alır ve Excel dosyasını temsil eden bir Workbook nesnesi döndürür. Çalışma kitabı yüklendikten sonra sayfa adlarına erişilebilir, etkin sayfa belirlenebilir ve sayfalar isimleri üzerinden seçilebilir.

Çalışma kitabı yüklenirken dosya adının hangi dizine göre çözümlendiği önemlidir. Dosya farklı bir konumdaysa Python onu bulamayabilir. Bu nedenle örneklerin sorunsuz çalışması için dosyanın geçerli çalışma dizininde yer alması gerektiği vurgulanır.

Sayfalardan Hücreleri Alma ve Değer Okuma

Bir çalışma sayfası alındıktan sonra hücrelere "A1", "B1" gibi koordinatlarla erişilerek hücre nesnesi elde edilir ve içerik value özniteliği üzerinden okunur. Hücre nesnesi ayrıca row, column ve coordinate öznitelikleri ile satır, sütun ve koordinat bilgilerini verir. Bazı durumlarda openpyxl'nin özellikle tarih değerlerini metin yerine datetime türünde yorumlayabildiği ifade edilir.

Sütunların harflerle gösterilmesi, özellikle Z'den sonra AA, AB gibi adlandırmalar başladığında kodla işlem yaparken zorlayıcı olabilir. Bu nedenle hücrelere cell() yöntemi kullanılarak satır ve sütun numaralarıyla erişmek mümkündür. Bu yaklaşımda indeksleme 0'dan değil 1'den başlar ilk satır ve ilk sütun numarası 1 kabul edilir.

Sayfadaki verinin boyutunu anlamak için satır ve sütun sayılarıyla ilgili özniteliklerden yararlanılır. Böylece kullanılan veri aralığı belirlenir ve okuma işlemi satır satır ya da sütun sütun olacak şekilde planlanır. Büyük tablolarda veriyi toplu okumak gerektiğinde satır ve sütunlar üzerinde dolaşmaya izin veren yöntemler devreye girer. Belirli bir aralıktaki hücreleri listelemek, satırları ardışık biçimde okumak, yalnızca değerleri almak ve Python tarafında işlemek bu sürecin temelini oluşturur. Bu bölümde koordinat temelli erişim ile sayısal indeks temelli erişim arasındaki farklara ve hangi durumda hangi yöntemin daha kullanışlı olduğuna değinilir.

Excel Belgelerini Yazma

Excel dosyalarıyla çalışmanın yalnızca okuma değil, aynı zamanda yeni dosyalar üretme ve mevcut dosyaları güncelleme boyutu da vardır. Bu bölüm, openpyxl ile Excel belgeleri oluşturma, sayfa ekleme ya da var olan sayfaları kullanma, hücrelere değer yazma ve değişiklikleri kaydetme süreçlerine odaklanır.

Excel Belgeleri Oluşturma Ve Kaydetme

Yeni bir Excel belgesi üretmek için openpyxl'nin çalışma kitabı oluşturma yaklaşımı kullanılır. Yeni bir Workbook nesnesi oluşturulduğunda genellikle varsayılan bir sayfa ile başlanır. İhtiyaç halinde yeni çalışma sayfaları eklenebilir, sayfa adları değiştirilebilir ve sayfalar arası düzen kurulabilir.

Belge üretiminde veriler hücrelere yazıldıktan sonra sonuçların kalıcı olması için dosyanın kaydedilmesi gerekir. Kaydetme işlemi, yapılan değişiklikleri .xlsx formatında bir dosyaya yazar. Böylece Python ile üretilen çalışma kitabı daha sonra Excel veya uyumlu bir uygulama ile açılıp kullanılabilir.

Mevcut bir dosyada değişiklik yapmak için önce dosya yüklenir, ilgili sayfa seçilir, hücre değerleri güncellenir ve çalışma kitabı tekrar kaydedilir. Bu akış, otomasyon senaryolarında aynı işlemin farklı dosyalara uygulanmasına da imkân sağlar.

Hücrelere Değer Yazma

Hücrelere değer yazma, hücre nesnesinin value özniteliğine yeni bir içerik atanmasıyla yapılır. Bu işlem koordinat kullanılarak ya da cell() yöntemiyle satır–sütun numaraları verilerek gerçekleştirilebilir. Böylece belirli hücrelere sabit değerler yazmak, hesaplama sonuçlarını hücrelere aktarmak veya döngüyle çok sayıda hücreyi güncellemek mümkün olur.

Hücre yazma işlemleri tek tek atamalarla sınırlı değildir. Satır bazında veri yerleştirme veya belirli bir aralığı toplu biçimde doldurma gibi ihtiyaçları da kapsar. Yapılan değişikliklerin dosyaya yansıması için kaydetme adımının gerekli olduğu hatırlatılır.

Hücrelerin Yazı Tipi Stilini Ayarlama

Excel belgelerinde yalnızca verinin kendisi değil, verinin okunabilir biçimde sunulması da önemlidir. Bu bölümde hücrelerin yazı tipi stilini ayarlamaya yönelik işlemler ele alınır. Yazı tipi, kalınlık, eğiklik, altı çizili olma gibi tipografik özellikler belirli sınıflar ve öznitelikler üzerinden yönetilebilir. Biçimlendirme tek bir hücreye uygulanabileceği gibi bir hücre aralığına da uygulanabilir. Bu yaklaşım, rapor üretiminde başlıkları vurgulamak, önemli değerleri belirginleştirmek ve tabloları daha düzenli göstermek için kullanılır. Böylece Python ile üretilen Excel çıktıları sadece veri içermekle kalmaz, sunum açısından daha anlaşılır bir yapıya da kavuşur.

Formüller

Excel'in güçlü yanlarından biri hücreler arası ilişkileri formüllerle kurabilmesidir. openpyxl ile çalışırken hücrelere yalnızca değer değil, formül de yazılabilir. Bu sayede toplam, ortalama gibi hesaplamalar Excel'in kendi hesaplama mekanizmasına bırakılır ve dosya açıldığında sonuçlar hesaplanır.

Formül yazmada temel yaklaşım, hedef hücrenin value alanına Excel formül söz dizimini yerleştirmektir. Böylece hücre, doğrudan bir değer taşımak yerine hesaplanan bir hücre gibi davranır. Bu yöntem, otomatik raporlama senaryolarında formüllerin doğru hücrelere programatik olarak yerleştirilmesi açısından kullanışlıdır.

Grafikler

Verileri daha hızlı yorumlayabilmek için grafikler önemli bir görselleştirme aracıdır. Bu bölümde openpyxl ile Excel çalışma kitaplarında grafik oluşturma mantığı ele alınır. Grafik üretimi genellikle veri aralığını belirleme, grafik türünü seçme ve grafiği sayfaya ekleme adımlarını içerir.

Grafik oluştururken hangi hücre aralığının veri kaynağı olacağı ve kategori etiketlerinin nasıl kullanılacağı önem taşır. Böylece grafik, tablodaki sayısal değerleri görsel bir özet hâline getirir. Sonuç olarak Python ile otomatik üretilen Excel dosyaları yalnızca tablo değil, görsel sunum bileşenleri içeren bir rapor çıktısı olarak da hazırlanabilir.

WORD VE PDF DOSYALARININ YAZILIM ARACILIĞIYLA YÖNETİMİ

Bu ünitede Word ve PDF dosyalarının Python programlama dili kullanılarak nasıl yönetileceği ele alınmıştır. Günümüzde birçok akademik ve kurumsal süreç dijital belgeler üzerinden yürütülmektedir. Raporlar, resmi yazışmalar, proje dokümanları ve arşiv kayıtları elektronik ortamda hazırlanmakta ve saklanmaktadır. Bu belgelerin manuel olarak hazırlanması zaman kaybına ve biçimsel tutarsızlıklara yol açabilmektedir. Bu nedenle belge işlemlerinin yazılım aracılığıyla gerçekleştirilmesi önemli bir avantaj sağlamaktadır.

Bu ünitede Word belgeleri için python-docx kütüphanesi, PDF belgeleri için ise PyPDF2 kütüphanesi kullanılmıştır. Uygulamalar Google Colab ortamında gerçekleştirilmiş ve dosya yükleme ile indirme süreçleri adım adım açıklanmıştır. Amaç, öğrencilerin hem belge üretmeyi hem de dosya işleme mantığını kavramalarını sağlamaktır.

Word Belgelerinin Yönetimi

Word belgeleri düzenlenebilir metin dosyalarıdır. Metin, tablo, görsel ve biçimlendirme bilgilerini bir arada barındırırlar. Python ile Word belgesi oluşturmak için python-docx kütüphanesi kullanılmaktadır.

Ünite kapsamında öncelikle yeni bir Word belgesi oluşturma işlemi gösterilmiştir. Daha sonra belgeye paragraf ekleme işlemi açıklanmıştır. Paragraf yapısı ve metin biçimlendirme üzerinde durulmuştur. Özellikle bir paragraf içerisindeki belirli kelimelerin kalın, italik veya altı çizili yapılabileceği örneklerle gösterilmiştir. Böylece belge içerisinde vurgulama yapılabileceği açıklanmıştır.

Word belgelerinde başlık yapısı da önemli bir konudur. Bu nedenle farklı seviyelerde başlık ekleme işlemi ele alınmıştır. Böylece belgede düzenli ve hiyerarşik bir yapı oluşturulabileceği gösterilmiştir. Ayrıca sayfa sonu ekleme işlemi ile belgeyi bölümlere ayırma süreci anlatılmıştır.

Tablo oluşturma da ünitede yer alan önemli konulardan biridir. Tabloya satır ve sütun ekleme, başlık satırı oluşturma ve veri girme işlemleri uygulamalı olarak gösterilmiştir. Bu sayede otomatik rapor üretiminde tablo kullanımının nasıl yapılacağı açıklanmıştır.

Belgeye görsel ekleme işlemi de ele alınmıştır. Görselin boyutunun ayarlanması ve belge içerisinde düzenli şekilde yerleştirilmesi süreci açıklanmıştır. Bu uygulama özellikle rapor ve sunum belgeleri için önemlidir.
Son olarak mevcut bir Word belgesinin açılması ve içerisindeki metnin okunması gösterilmiştir. Bu işlem, var olan belgelerin analiz edilmesi veya düzenlenmesi gereken durumlarda kullanılmaktadır.

PDF Belgelerinin Yönetimi

PDF dosyaları sabit düzenli belge formatlarıdır. Metin dosyalarından farklı olarak ikili yapıda saklanırlar. Bu nedenle özel kütüphaneler kullanılarak işlenirler. Bu ünitede PDF işlemleri için PyPDF2 kütüphanesi kullanılmıştır.

İlk olarak PDF dosyasından metin çıkarma işlemi ele alınmıştır. Belirli bir sayfanın metni okunmuş ve ekrana yazdırılmıştır. Bu işlem belge içeriğinin analiz edilmesi açısından önemlidir.

PDF belgelerinde sayfa düzenleme işlemleri de açıklanmıştır. Belirli sayfaların seçilerek yeni bir PDF oluşturulması, sayfa sırasının değiştirilmesi ve iki PDF dosyasının birleştirilmesi uygulamalı olarak gösterilmiştir. Bu işlemler belge düzenleme ve arşivleme süreçlerinde sıkça kullanılmaktadır.

Dosya güvenliği konusu da ele alınmıştır. PDF dosyasına şifre koyma ve şifreli PDF dosyasını açma işlemleri açıklanmıştır. Bu sayede belgelerin yetkisiz erişime karşı korunabileceği gösterilmiştir.

Ayrıca PDF dosyalarının okuma ve yazma modlarında açılması gerektiği teknik olarak açıklanmıştır. “ rb” ifadesi dosyayı ikili okuma modunda açmak için, “wb” ifadesi ise ikili yazma modunda açmak için

kullanılmaktadır. Bu ayrım, dosyaların doğru biçimde işlenmesi açısından önemlidir.

Genel Değerlendirme
Bu ünitede Word ve PDF belgelerinin Python ile yönetilmesi kapsamlı biçimde ele alınmıştır. Öğrencilerin belge oluşturma, düzenleme, biçimlendirme, tablo ve görsel ekleme, metin çıkarma, sayfa düzenleme ve şifreleme işlemlerini uygulayabilmeleri hedeflenmiştir.

Belge işlemlerinin otomatikleştirilmesi hem akademik hem de kurumsal ortamlarda önemli bir beceridir. Otomatik rapor üretimi, güvenli belge paylaşımı ve arşiv düzenleme gibi süreçler bu bilgi sayesinde daha hızlı ve güvenli şekilde gerçekleştirilebilir.

Bu ünitenin sonunda öğrencilerin dijital belge yönetimi konusunda temel teknik yeterliliğe sahip olmaları ve belge işlemlerini yazılım aracılığıyla sistemli bir şekilde gerçekleştirebilmeleri amaçlanmıştır.