The Pragmatic Programmers

Practical Programming

Third Edition

An Introduction to Computer Science Using Python 3.6

Paul Gries
Jennifer Campbell
Jason Montojo
edited by Tammy Coron

BÖLÜM 5

Seçimler Yapmak

Sıla GÜCEM

Bu bölüm programlamanın bir başka temel kavramını tanıtmaktadır. Bunu, programımızın birlikte çalıştığı verilere bağlı olarak farklı davranmasını istediğimizde yaparız. Örneğin, bir çözeltinin asidik veya bazik olmasına veya kullanıcının yerleşik işlev girişindeki bir çağrıya yanıt olarak evet veya hayır yazmasına bağlı olarak farklı şeyler yapmak isteyebiliriz. Bu bölümde kontrol akışı ifadeleri olarak adlandırılan seçimler yapmaya yönelik ifadeleri tanıtacağız [çünkü bunlar bilgisayarların programları yürütme şeklini kontrol eder]. Bu ifadeler doğruluk ve yanlışlığı temsil etmek için kullanılan bir Python türü içerir. Daha önce gördüğümüz tamsayılar, noktalı sayılar ve dizgilerden farklı olarak, bu türün yalnızca iki değeri ve üç işlevi vardır.

Bir boole Türü

Python ' da bool ("e" olmadan) adında bir tür vardır. int ve float'ın aksine milyarlarca olası değere sahipken, bool' un yalnızca iki değeri vardır: Doğru ve Yanlış. Doğru ve Yanlış tıpkı 0 ve -43,7 sayıları gibi değerlerdir.

George Boole

1840'larda matematikçi George Bolle, klasik matematik kurallarının mantığı, yalnızca iki doğru değeri kullanılarak tamamen matematiksel biçimde ifade edilebilir ve yalnış. Bir yüzyıl sonra, Claude Shannon (bilgi teorisinin mucidi) şunu verketti Boole'un çalışmasının elektromekanik telefon tasarımını optimize etmek içib Boole mantığının kullanılmasına yol açtı.

Boole'un çalışmalarının onuruna, çoğu modern programlama dili neyin doğru neyin yanlış olduğunu takip etmek için.

Boolean Operatorleri

Yalnızca temel Boolean operatr vardır: and, or ve not, ve, veya, ve değil. Not tekli

bir operatördür: tek bir değere uygulanır , tıpkı olumsuzlama gibi.
Orjinal ifade True üretmiyorsa içeren bir ifade değeri False ise False üretir ve orjinal değer True ise False üretir:

```
>>> not True
False
>>> not False
True
```

Önceki örnekte, not True yerine False kullanabiliriz. Not'u doğrudan bir Boolean'a uygulamak yerine değerine değil, genellikle bir Boole değişkenine veya daha karmaşık bir değişkene uygulanmaz Boolean ifadesi. Aynı durum aşağıdaki Boolean örnekleri için de geçerlidir ve veya operatörleri, bu nedenle bunlrı Boolean sabitine uygulasak da aşağıdaki örneklerede, tipik olartak nasıl kullanıldıklarına dair bir örnek vereceğiz bu bölümün sonunda.
Ve ikili bir operatördür. Her iki işlenen de True ise True üretir ve aksi takdirde yanlış olur:

```
>>> True and True
True
>>> False and False
False
>>> True and False
False
>>> False and True
False
```

or aynı zamanda bir ikili işleçtir. İşlenenlerden biri True ise True üretir ve yalnızca her ikisi de False ise False üretir:

```
>>> True or True
True
>>> False or False
False
>>> True or False
True
>>> False or True
True
```

Bu tanıma kapsayıcı denir veya der iki olasılığa da izin verdiği için her ikisi de. İngilizce

'd eveya kelimesi bazen özel veya olarak da kullanılır. Örneğin biri "Pizza ya da tandır

tavuğu yiyebilirsin" derse,

Muhtemelen her ikisine de sahip olabileceğiniz anlamına gelmez. İngilizceden farklı olarak (ancak çoğu gibi) programlama dilleri, Python her zaman veya kapsayıcı olarak yorumlanır.

ÖZEL VEYA İFADE OLUŞTURMA

Özel bir veya istiyorsanız, bunun için bir Boolean ifadesi oluşturmanız gerekir. Bu ifadenin geliştirilmesi yoluyla yürüyeceğiz.

Diyelim ki iki Boole değişkeniniz var, b1 ve b2: Yalnızca bunlardan birinin Doğru olmas ı durumunda Doğru olarak değerlendirilir şeklinde bir ifade istiyorsunuz. b1' in değerlendirilmesi ve b2 'nin değerlendirilmesi b1 Doğru ve b2 Yanlış ise Doğru üretecektir. Benzer şekilde, b1 Doğru ve b 1 Yanlış ise Doğru'yu üretir.

Bu ifadelerin her ikisinin de doğruyu üretmesi mümkün değildir. Ayrıca, eğer b1 ve b2 her ikiside True veya her ikiside False ise, her iki ifade de False olarak değerlendirilecektir. Bu nedenle sunları yapabilirsiniz , iki ifadeyi bir veya ile birleştirin:

```
>>> b1 = False  >>> b2 = False
>>> (b1 and not b2) or (b2 and not b1)
False
>>> b1 = False
>>> b2 = True
>>> (b1 and not b2) or (b2 and not b1)
True
>>> b1 = True
>>> b2 = False
>>> (b1 and not b2) or (b2 and not b1)
True
>>> b1 = True
>>> b2 = True
>>> (b1 and not b2) or (b2 and not b1)
False
```

Birkaç sayfa sorna çok daha basit bir versiyonunu göreceğiz.

Daha önce Boolean operatörlerinin genellikle Boolean'a uygulandığından bahsetmiştik Boolean sabiti yerine ifadeler. Eğer ifade etmek istersek "Bu değil soğuk ve rüzgarlı" ifadesini Boolean değerinde atıfta bulunan iki değişken (soğuk ve rüzgarlı) kullanarak yazınız, öncelikle muğlak İngilizce ifadenin ne anlama geldiğine karar vermeliyiz: bu ifade Soğuk değil ama aynı zamanda rüzgarlı mı , yoksa hem soğuk değil hemde rüzgarlı değil mi ? Herbir alternatif için doğruluk tablosu Tablo 5, Boole Opratörleri'nde gösterilmiştir. Sayfa 80 ve aşşağıdaki kod parçacığı çevrilmiş olarak nasıl göründüklerini göstermektedir.

Python' a :

```
>>> cold = True
>>> windy = False
>>> (not cold) and windy
False
>>> not (cold and windy)
True
```

cold	windy	cold and windy	cold or windy	(not cold) and windy	not (cold and windy)
True	**True**	True	True	False	False
True	**False**	False	True	False	True
False	**True**	False	True	True	True
False	**False**	False	False	False	True

Tablo 5—Boolean Operatorleri

Diğer Dillerde Boolean Operatorleri

C veya Java gibi başka bir dili zaten biliyorsanız, &&'ye alışkın olabilirsiniz. for || for or, ! for not. İfadeleri Python'da Çalışmaz, ancak fikir aynıdır.

İlişkisel İşlemciler

Daha önce True ve False'ın birer değer olduğunu söylemiştik. Tipik oalrak bu değerler do ğrudan ifadeler halinde yazılır, daha ziyadde ifaeler halinde oluşturulur. İfadeler bunu yapmanın en yaygın yolu, bir ilişkisel yöntem kullanarak karşılaştırma yapmak için operatörler kullanılır. Örneğin , 3<5 ilişkisel operatör < kullanılarak yapılan bir karşılaştırmadır.

Tablo 6'da, Python kullanmaya alışık olduğunuz tüm operatörlere sahiptir. Bazılar bir yerine iki karakter kullanılarak gösterilir, örenğin ≤ yerine <=.

Symbol	Operation
>	Büyüktür
<	Küçüktür
>=	Büyük eşittir
<=	Küçük esittir
= =	**Eşittir**
!=	Eşit Değildir

Tablo 6—İlişkisel ve Eşitlik Operatorleri

En önemli temsil kuralı Python'un eşitlik için == kullanmasıdır. Çünkü = atma için kullanılır. Şu durumlarda x=3 yazmaktan kaçının x değişkeninin üçe eşit olup olmadığını kontrol etmek istiyorsunuz: x==3 .

Tüm ilişkisel operatörler ikili operatörlerdir: iki değeri karşılaştırırlar ve uygun şekilde Doğru veya Yanlış üretir. Büyüktür (>) ve küçüktür (<) operatörleri aşağıdaki gibi çalışır:

```
>>> 45 > 34
True
>>> 45 > 79
False
>>> 45 < 79
True
>>> 45 < 34
False
```

Tam sayıları kayan noktalı sayılarla herhangi bir ilişkisel sayıyla karşılaştırabiliriz. otomatik olarak kayan noktaya dönüştürülür. Bu, tıpkı 14'ü 23,3 eklediğimizde olduğu gibi

```
>>> 23.1 >= 23
True
>>> 23.1 >= 23.1
True
>>> 23.1 <= 23.1
True
>>> 23.1 <= 23
False
```

Aynı şey "eşittir" ve "eşit değildir" için de geçerlidir

```
>>> 67.3 == 87

False
>>> 67.3 == 67
False
>>> 67.0 == 67
True
>>> 67.0 != 67
False
>>> 67.0 != 23
True
```

Tabii ki, iki sayıyı karşılaştırmak pek mantıklı değil. Karşılaştırmanın sonucunu da önceden bilmeniz gerekir. Bu nedenle ilişkisel operatörler neredeyse her zaman değişkenler içerir, bunun gibi:

```
>>> def is_positive(x: float) -> bool:
...     """Return True iff x is positive.
...
...     >>> is_positive(3)
...     True
...     >>> is_positive(-4.6)
...     False
...     """
...     return x > 0
...
>>> is_positive(3)
True
```

```
>>> is_positive(-4.6)
False
>>> is_positive(0)
False
```

Idas Bu döküman dizisinde " if and only if" anlamına gelen "iff" kısaltmasını kullanıyoruz. Eş değer bir ifade "tam olarak ne zaman" dır. Tür sözleşmesi şunu belirtir fontsiyonu bir bool döndürecektir. Dökümantasyonun hangi koşullar altında olduğunu açıklar. Bu koşullar yerine getirilmediğinde işlevi False değerine döndürülecektir.

*Building a*n Exclusive or'dan yazabiliriz sayfa 79'da cok daha basit bir şekilde var.

```
b1 != b2
```

Exclusive or, b1 be b2 ' den tam olarak birinin True olması gerektiği anlamına gelir. E ğer b1 Doğru ise b2 olamaz, yada tam tersi.

Karşılaştırmaların Birleştirilmesi

Şimdi üç tür operatör gördük: aritetik (+, -, vb.), Boolean (and, or, and not), ve kişisel (<, ==, vb.).

İşte bunlarıebirleştirmek için kurallar:

- Aritmatik operatörler ilişkisel operatörlerden daha yüksek önceliğe sahiptir.

Örneğin , + ve / <veya> 'den önce değerlendirilir.

- İlişkisel operatörlerde Boole operatörlerinden daha yüksek önceliğe sahiptir. Örneğin, karşılaştırmalar and, or ve not öğeleriden önce değerlendirilir.
- Tüm ilişkisel operatörler aynı önceliğe sahiptir.

Bu kurallar, 1+3 > 7 ifadesinin (1+3) >7 olarak değerlendirileceği anlamına gelir. 1+ (3>7) gibi. Bu kurallar aynı zamanda parantezleri genellikle atayabileceğiniz anlamına gelir karmaşık ifadelerde:

```
>>> x = 2  >>> y = 5  >>> z = 7  >>> x < y and y < z
True
```

Yine de parantezleri koymak genellikle iyi bir fikirdir, çünkü göz alt ifadeleri bulur ve sıralamayı herkese açıkça iletir:

```
>>> x = 5
>>> y = 10
>>> z = 20
>>> (x < y) and (y < z)
True
```

Matematikte bir değerin belirli bir değer aralığında olup olmadığını kontrol etmek çok yaygındır.ÿAralık başka bir deyişle diğer iki değer arasında olduğunu gösterir. Şunu yapabilrisiniz.

Python'da karşılaştırmaları ile bileştirerek:

```
>>> x = 3  >>> (1 < x) and (x <= 5)
True
>>> x = 7
>>> (1 < x) and (x <= 5)
False
```

Ancak bu durumla o kadar sık karşılışır ki Python karşılaştırmaları zincirlemenize izin verir:

```
>>> x = 3
>>> 1 < x <= 5
True
```

Çoğu kombinasyon beklediğiniz gibi çalışır, ancak bazı durumlarda korkutur:

```
>>> 3 < 5 != True
True
>>> 3 < 5 != False
True
```

Bu ifadelerin her ikisinin de Doğru olması mümkün görünmemektedir. Ancak, ilk biri buna eşdeğerdir:

`(3 < 5) and (5 != True)` İkincisi ise buna eşdeğerdir:

`(3 < 5) and (5 != False)`

5 ne Doğru ne de Yanlış olduğundan, her ifadenin ikinci yarısı Doğrudur, bu nedenle bir bütün olarak ifade de doğrudur.
Bu tür bir ifade , her ne kadar kötü bir fikir olsa de yasaldır.

Karşılaştırmaları yalnızca aşağıdaki şekillerde bir matematikçiye doğal gelecektir- başka bir deyişle, < ve <= birlikte veya > ve >= birlikte, başka bir şey değil. Eğer bir dürtü hissedereniz başka bir şey yapın, direnin. Basit karşılaştırmalar kullanın ve bunları ve ile birleştirin. Kodunuzu okunabilir tutmak için , paranter kullanmak da iyi bir fikirdir. Yazdığınız ifadenin tam olarak aç ık olmadığını düşündüğünüzde.

Boole İşlecleri ile Sayıları ve Dizileri Kullanma

Tamsayı kullanıldığında Python'un bir int'i float'a dönüştüreceğini daha önce kayan noktalı bir sayı içeren bir ifade görmüştük. Aynı doğrultuda, sayılar ve stringler Boolen operatörleri ile kullan ılabilir. Pyhton 0 ve 0.0 değerleri False ve diğer tüm sayıları Doğru olarak değerlendirir:

```
>>>  not 0  True
>>> not 1
False
>>> not 34.2
False
>>> not -87
False
```

Benzer şekilde, boş dize False olarak ve diğer tüm dizeler True olarak değerlendirilir:

```
>>> not ''

True
>>> not 'bad'
False
```

None da False olarak değerlendirilir. Genel olarak, Boole operatörlerini yalnızca Boolean değerleri.

Emre YÜCEL

Kısa Devre Değerlendirmesi (Short-Circuit Evaluation)

Python, and veya or içeren bir ifadeyi soldan sağa doğru değerlendirir. Yeterli bilgiye ulaştığında, değerlendirmeyi durdurur; bu, bazı operandlara (işlem yapılacak değerler) henüz bakılmamış olsa bile geçerlidir. Buna "kısa devre değerlendirmesi" denir.

Bir or ifadesinde, eğer ilk operand True (doğru) ise, ifadenin True olduğunu biliyoruz. Python da bunu bildiği için ikinci operandı değerlendirmeye bile almaz. Benzer ekilde, bir and ifadesinde, eğer ilk operand False (yanlış) ise, ifadenin False olduğunu biliyoruz. Python bunu da bildiği için ikinci operandı değerlendirmez.

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Şimdi bu ifadeyi or ifadesinin ikinci operandı olarak kullanıyoruz:

```
>>> (2 < 3) or (1 / 0)
True
```

İlk operand True (doğru) değerini ürettiği için, ikinci operand değerlendirilmez, bu yüzden bilgisayar gerçekten de sıfıra bölme işlemi yapmamış olur.

Tabii ki, bir or ifadesinde ilk operand False (yanlış) olduğunda, ikinci operandın değerlendirilmesi gerekir. Ayrıca, bir and ifadesinde ilk operand True (doğru) olduğunda, ikinci operandın da değerlendirilmesi gerekir.

Dizilerin Karşılaştırılması (Comparing Strings)

Dizileri, sayıları karşılaştırdığınız gibi karşılaştırmak da mümkündür. Dizilerdeki karakterler, tamsayılarla temsil edilir: örneğin, büyük A 65 ile, boşluk 32 ile, küçük z ise 122 ile gösterilir. Bu kodlama "ASCII" olarak adlandırılır; "Amerikan Standart Bilgi Değişim Kodu"nun kısaltmasıdır. Bu sistemin bir özelliği, tüm büyük harflerin, tüm küçük harflerden önce gelmesidir; bu yüzden büyük Z, küçük a'dan daha küçüktür.

İki diziyi karşılaştırmanın en yaygın nedenlerinden biri, hangisinin alfabetik olarak önce geldiğine karar vermektir. Bu genellikle "sözlük sıralaması" veya "leksikografik sıralama" olarak adlandırılır. Python, dizilerin hangisinin büyük olduğunu belirlerken soldan sağa doğru karşılık gelen karakterleri karşılaştırır. Eğer bir dizinin karakteri, diğer dizinin karakterinden büyükse, ilk dizi ikinci diziden büyüktür. Eğer tüm karakterler aynıysa, iki dizi eşittir; eğer karşılaştırma yapılırken bir dizinin karakterleri bitiyorsa (başka bir deyişle, diğerinden daha kısaysa), o dizi daha küçüktür. Aşağıdaki kod parçası, birkaç karşılaştırmayı gösterir:

```
>>> 'A' < 'a'
True
>>> 'A' > 'z'
False
>>> 'abc' < 'abd'
True
>>> 'abc' < 'abcd'
True
```

Dizileri leksikografik olarak karşılaştıran operatörlere ek olarak, Python, bir dizinin başka bir dizinin içinde olup olmadığını kontrol eden bir operatör de sağlar.

```
>>> 'Jan' in '01 Jan 1838'
True
>>> 'Feb' in '01 Jan 1838'
False
```

Bu fikri kullanarak, kullanıcıdan bu formatta bir tarih girmesini isteyebiliriz ve o tarihin Ocak ayında olup olmadığını bildirebiliriz.

1. Bakınız https://en.wikipedia.org/wiki/ASCII.

```
>>> date = input('Enter a date in the format DD MTH YYYY: ')
Enter a date in the format DD MTH YYYY: 24 Feb 2013
>>> 'Jan' in date
False
>>> date = input('Enter a date in the format DD MTH YYYY: ')
Enter a date in the format DD MTH YYYY: 03 Jan 2002
>>> 'Jan' in date
True
```

in operatörü, ilk dizinin ikinci dizinin içinde bulunduğu durumlarda tam olarak True (doğru) döndürür. Bu işlem büyük/küçük harf duyarlıdır:

```
>>> 'a' in 'abc'
True
>>> 'A' in 'abc'
False
```

Boş dizi, her dizinin her zaman bir alt dizesidir:

```
>>> '' in 'abc'
True
>>> '' in ''
True
```

in operatörü, diğer türler için de geçerlidir; bununla ilgili diğer örnekleri bölüm 8, "Veri Koleksiyonlarını Listelerle Saklama" (sayfa 129) ve Bölüm 11, "Diğer Koleksiyon Türleriyle Veri Saklama" (sayfa 203) kısımlarında görebilirsiniz.

Hangi İfadelerin Çalıştırılacağını Seçme

Bir if ifadesi, programınızın davranışını bir koşula bağlı olarak değiştirmenizi sağlar. Bir if ifadesinin genel ekli öyledir:

```
if «condition»:

    «block»
```

Koşul, color != “neon green” veya x < y gibi bir ifadedir. (Bu ifadenin bir Boolean ifadesi olması gerekmez. Sayfada 84'te "Boolean Operatörleri ile Sayılar ve Diziler Kullanımı" bölümünde tartıştığımız gibi, Boolean olmayan değerler gerektiğinde True veya False olarak değerlendirilir.)

Fonksiyon gövdeleriyle olduğu gibi, bir if ifadesinin içindeki ifade bloğu da girintili olmalıdır. Hatırlatma olarak, Python için standart girinti dört boşluktur.

Eğer koşul doğruysa, bloktaki ifadeler çalıştırılır; aksi takdirde çalıştırılmaz. Fonksiyonlarda olduğu gibi, ifadeler bloğu, if ifadesine ait olduğunu göstermek için girintili olmalıdır. Eğer düzgün bir ekilde girinti yapmazsanız, Python bir hata verebilir veya daha kötüsü, yazdığınız kodu sevinçle çalıştırabilir.

Ancak bazı ifadeler düzgün bir ekilde girintilenmediği için niyet ettiğinizden farklı bir şey yapabilir. Bu bölümde her iki sorunu da kısaca inceleyeceğiz.

İşte pH seviyesine dayanan çözüm kategorileri tablosu:

pH Leveli	Cözüm Kategorisi
0–4	Güclü asit
5–6	Zayıf asit
7	Doğal
8–9	Zayıf baz
10–14	Güclü baz

Tablo 7— Çözüm Kategorileri

pH seviyesi kullanıcının belirttiği değere göre asidik olduğunda bir mesaj yazdırmak için bir if ifadesi kullanabiliriz:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 6.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
...
6.0 is acidic.
```

Klavye ile Bilgi Alma bölümünden (sayfa 73) hatırlarsanız, kullanıcı girişi yapılmadanönce bir dizi olan veriyi kayan noktalı bir sayıya dönüştürmemiz gerektiğini biliyoruz.Ayrıca burada, kullanıcıya ne yazması gerektiğini bildirmek için input fonksiyonuna birdizi geçirerek bir istem sağlıyoruz; Python bu dizi ile kullanıcıya bilgi verir.

Eğer koşul yanlışsa, bloktaki ifadeler çalıştırılmaz:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 8.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
...
>>>
```

Eğer bloğu girintilemezsek, Python bize bilgi verir:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 6
>>> if ph < 7.0:
... print(ph, "is acidic.")
  File "<stdin>", line 2
    print(ph, "is acidic.")
        ^
```

IndentationError: beklenen bir girintili blok

Bir blok kullandığımız için, koşul doğruysa yalnızca çalıştırılan birden fazla ifade olabilir:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 6.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
...     print("You should be careful with that!")
...
6.0 is acidic.
You should be careful with that!
```

Blokta ilk satırı girintilediğimizde, Python yorumlayıcısı, blok bitene kadar istemini ...şeklinde değiştirir; blokun sonu ise boş bir satırla belirtilir:

```
>>> ph = float(input('Enter the pH level: '))

Enter the pH level: 8.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
...
>>> print("You should be careful with that!")
You should be careful with that!
```

Eğer bloktaki kodu girintilemezsek, yorumlayıcı şikayet eder:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 8.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
... print("You should be careful with that!")
  File "<stdin>", line 3
    print("You should be careful with that!")
        ^
```

SyntaxError: geçersiz sözdizimi

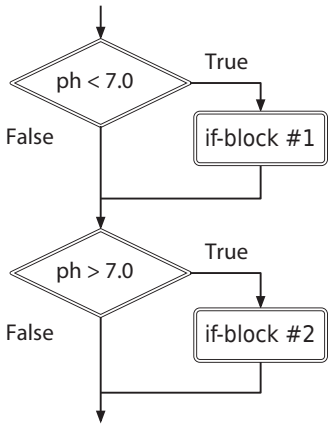
Eğer program bir dosyada bulunuyorsa, boş bir satıra ihtiyaç yoktur. Girinti sona erdiğianda, Python bloğun da sona erdiğini varsayar. Bu nedenle, bu durum geçerlidir:

```
ph = 8.0  if ph < 7.0:



    print(ph, "is acidic.")
print("You should be careful with that!")
```

Elbette, bazen tek bir kararın yeterli olmadığı durumlarla karşılaşırız. Birden fazlakriterin incelenmesi gerektiğinde, bunu ele almanın birkaç yolu vardır. Bunlardan biri,birden fazla if ifadesi kullanmaktır. Örneğin, öyle yapabiliriz:pH seviyesinin asidik veya bazik olup olmadığına göre farklı mesajlar yazdırabiliriz (eğertam olarak 7 ise, o zaman nötrdür ve kodumuz hiçbir ey yazdırmaz):

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 8.5
>>> if ph < 7.0:
...     print(ph, "is acidic.")
...
>>> if ph > 7.0:
...     print(ph, "is basic.")
...
8.5 is basic.
>>>
```

İşte Python'un if ifadelerini nasıl çalıştırdığını gösteren bir akış şeması. Elmaslarkoşulları, oklar ise bu koşulların değerlendirilmesi sonucuna göre hangi yoldanilerleyeceğimizi gösterir:

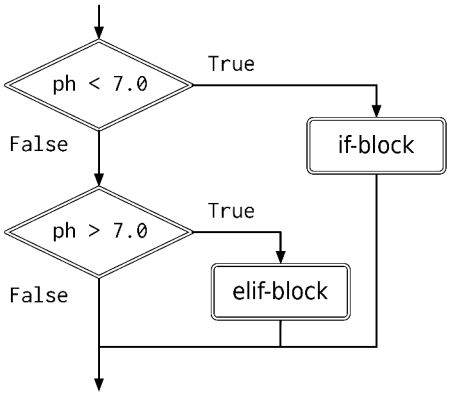


Her iki koşulun da her zaman değerlendirildiğine dikkat edin; oysa yalnızca bir blokçalıştırılabilir.

Her iki durumu birleştirmek için, elif anahtar kelimesini kullanarak başka bir koşul/blokçifti ekleyebiliriz (elif, "else if" anlamına gelir); her koşul/blok çifti bir "clause" (kısım)olarak adlandırılır:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 8.5
>>> if ph < 7.0:
...     print(ph, "is acidic.")
... elif ph > 7.0:
...     print(ph, "is basic.")
...
8.5 is basic.
>>>
```

İkisi arasındaki fark, elif'in yalnızca yukarısındaki if koşulu False (yanlış) olarak değerlendirildiğinde kontrol edilmesidir. İşte bu kod için bir akış şeması:



Bu akış şeması, eğer ilk koşul True (doğru) olarak değerlendirilirse, ikinci koşulunatlandığını gösterir.Eğer pH tam olarak 7.0 ise, hiçbir kısım eşleşmez, bu nedenle hiçbir ey yazdırılmaz:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 7.0
>>> if ph < 7.0:
...     print(ph, "is acidic.")
... elif ph > 7.0:
...     print(ph, "is basic.")
...
>>>
```

pH örneğinde, iki if ifadesiyle aynı şeyi başardık; bu, bir if/elif ile elde ettiğimizle aynıydı.

Bu her zaman böyle değildir; örneğin, ilk if'in gövdesi, ikinci koşulda kullanılan birdeğişkenin değerini değiştirirse, bunlar eşdeğer değildir. İşte iki if ile olan versiyonu:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 6.0
>>> if ph < 7.0:
...     ph = 8.0
...
>>> if ph > 7.0:
...     print(ph, "is acidic.")
...
8.0 is acidic.
```

Ve işte bir if/elif ile olan versiyonu:

```
>>> ph = float(input('Enter the pH level: '))
Enter the pH level: 6.0
>>> if ph < 7.0:
...     ph = 8.0
>>> elif ph > 7.0:
...     print(ph, "is acidic.")
...
>>>
```

Bir kural olarak, eğer iki koşul birbirine bağlıysa, iki if yerine if/elif kullanın.
Bir if ifadesinin ardından birden fazla elif kuralı gelebilir. Bu daha uzun örnek, birkimyasal formülü İngilizceye çevirir:

```
>>> compound = input('Enter the compound: ')
Enter the compound: CH4
>>> if compound == "H2O":
...     print("Water")
... elif compound == "NH3":
...     print("Ammonia")
... elif compound == "CH4":
...     print("Methane")
...
Methane
>>>
```

Sayfa 90'daki kodda gördüğümüz gibi, eğer bir if/elif zincirindeki koşullardan hiçbirisağlanmıyorsa, Python ilişkilendirilmiş hiçbir bloğu çalıştırmaz.

Ancak bu her zaman istediğimiz gibi olmaz. Çeviri örneğimizde, programımızın bileşiğitanımadığında bile bir ey yazdırmasını istiyoruz.

Bunu yapmak için, zincirin sonuna bir else kuralı ekleriz:

```
>>> compound = input('Enter the compound: ')
Enter the compound: H2SO4
>>> if compound == "H2O":
...     print("Water")
... elif compound == "NH3":
...     print("Ammonia")
... elif compound == "CH4":
...     print("Methane")
... else:
...     print("Unknown compound")
...
Unknown compound
>>>
```

Bir if ifadesinin en fazla bir else kuralı olabilir ve bu kural ifadenin son kuralı olmalıdır.Else ile ilişkili bir koşul olmadığını unutmayın:

```
if «condition»:
    «if_block»
else:
    «else_block»
```

Mantıksal olarak, bu kod, bu kodla aynıdır (ilk biçimde koşul yalnızca bir kez değerlendirilirken, ikinci biçimde iki kez değerlendirilir):

```
if «condition»:
    «if_block»
if not «condition»:
    «else_block»
```

Yahya Cemal AKYÜZ

İç İçe İf Deyimleri

Bir if ifadesinin bloğu herhangi bir Python ifadesini içerebilir, bu da başka if ifadelerini de içerebileceği anlamına gelir. Bir if ifadesinin içinde başka bir if ifadesi bulunmasına iç içe if (nested if) ifadesi denir.

```
value = input('Enter the pH level: ')
if len(value) > 0:
    ph = float(value)
    if ph < 7.0:
        print(ph, "is acidic.")
    elif ph > 7.0:
        print(ph, "is basic.")
    else:
        print(ph, "is neutral.")
else:
    print("No pH value was given!")
```

Bu durumda, kullanıcıdan bir pH değeri girmesini isteriz ve bu değeri başlangıçta bir string olarak alırız. lk veya dış if deyimi, kullanıcının bir ey yazıp yazmadığını kontrol eder, bu da pH değerini iç if deyimiyle inceleyip incelemeyeceğimizi belirler. (Eğer kullanıcı bir sayı girmediyse, float(value) fonksiyon çağrısı bir ValueError hatası üretecektir.)

İç içe if deyimleri bazen gerekli olabilir, ancak karmaşıklaşabilir ve anlaşılması zor hale gelebilir. Bir deyimin ne zaman yürütüleceğini açıklamak için koşulları zihinsel olarak birleştirmemiz gerekir; örneğin, print(ph, "asidik.") ifadesi yalnızca, value'nin referans verdiği string'in uzunluğu 0'dan büyükse ve pH < 7.0 ifadesi de True (doğru) olarak değerlendirilirse çalıştırılır (kullanıcının bir sayı girdiğini varsayarsak).

Bir Boolean fade Değerlendirmesinin Sonuçlarını Hatırlama

Aşağıdaki kod satırına bir göz atın ve x'e hangi değerin atandığını tahmin edin:

```
>>> x = 15 > 5
```

"! Bu durumda, x'e atanan değer True olacaktır çünkü 15, 5'ten büyüktür. !"

Eğer True dediyseniz, doğru söylediniz: 15, 5'ten büyüktür, bu yüzden karşılaştırma True sonucunu üretir ve bu da diğer değerler gibi bir değişkene atanabilir.

Bunu yapmanız gereken en yaygın durum, karar tablolarını yazılıma çevirirken ortaya çıkar. Örneğin, yaş ve vücut kitle indeksi (BMI) temel alınarak bir kişinin kalp hastalığı riskini hesaplamak istiyorsunuz:

		Age	
		<45	≥45
BMI	<22.0	Low	Medium
	≥22.0	Medium	High

Bunu uygulamanın bir yolu iç içe if deyimlerini kullanmaktır:

```
if age < 45:
    if bmi < 22.0:
        risk = 'low'
    else:
        risk = 'medium'
else:
    if bmi < 22.0:
        risk = 'medium'
    else:
        risk = 'high'
```

bmi < 22.0 ifadesi birkaç kez kullanılıyor. Bu kodu basitleştirmek için, her Boolean ifadesini bir kez değerlendirip, bu ifadelerin ürettiği değerleri referans alacak değişkenler oluşturabilir ve bu değişkenleri birden fazla kez kullanabiliriz.

```
young = age < 45
slim = bmi < 22.0
if young:
    if slim:
        risk = 'low'
    else:
        risk = 'medium'
else:
    if slim:
        risk = 'medium'
    else:
        risk = 'high'
```

Bunu iç içe kullanmadan da şu şekilde yazabiliriz:

```
young = age < 45
slim = bmi < 22.0
if young and slim:
    risk = 'low'
elif young and not slim:
    risk = 'medium'
elif not young and slim:
    risk = 'medium'
elif not young and not slim:
    risk = 'high'
```

İç içe geçirme kullanıp kullanmadığınıza bakılmaksızın, mantıksal değişkenlere (genç ve zayıf) anlamlı isimler vermek, kodu daha anlaşılır hale getirmeye yardımcı olur.

Booleans Hakkında Öğrendin: True mu, False mi?

Bu bölümde unları öğrendiniz:

- Python, doğru ve yanlış olanı temsil etmek için Boolean değerler, yani True ve False kullanır. Programlar, bu değerleri not, and, ve or gibi üç operatörle birleştirebilir.
- Boolean operatörleri sayısal değerlere de uygulanabilir. 0, 0.0, boş string (""), ve None False olarak kabul edilir; diğer tüm sayısal değerler ve stringler True olarak değerlendirilir. Boolean operatörlerini Boolean olmayan değerlere uygulamaktan kaçınmak en iyisidir.
- Eşittir ve küçüktür gibi karşılaştırma operatörleri değerleri karşılaştırır ve Boolean sonuçlar üretir.
- Farklı operatörler bir ifadede birleştirildiğinde, öncelik sırası en yüksekten en düşüğe doğru: aritmetik, karşılaştırma (relational), ve Boolean'dır.
- if deyimleri yürütme akışını kontrol eder. Fonksiyon tanımlamaları gibi, if deyimlerinin gövdeleri ve elif ve else kısımlarının gövdeleri de girintilidir.

Alıştırmalar

Kendi başınıza denemeniz için bazı alıştırmalar:

Çözümler: http://pragprog.com/titles/gwpy3/practical-programming

1. Her bir ifadenin ürettiği değeri bulun. Cevaplarınızı Python'a yazarak doğrulayabilirsiniz.

   a. True and not False

   b. True and not false (Dikkat: "false" küçük harflerle yazılmıştır.)

   c. True or True and False

   d. not True or not False

   e. True and not 0

   f. 52 < 52.3

   g. 1 + 52 < 52.3

   h. 4 != 4.0

2. x ve y değişkenleri Boolean değerleri veriyor.

   a. Her iki değişken de True olduğunda True döndüren bir ifade yazın.

   b. x False olduğunda True döndüren bir ifade yazın.

   c. Değişkenlerden en az biri True olduğunda True döndüren bir ifade yazın.

3. full ve empty değişkenleri Boolean değerleri verir. Şu ifadeyi yazın: En fazla bir değişken True olduğunda True döndüren bir ifade.

4. Bir otomatik vahşi yaşam kamerasının ışık seviyesi 0.01 lux'tan düşükse veya sıcaklık donma noktasının üzerindeyse açılmasını istiyorsunuz, ancak her iki koşul da doğruysa açılmamalıdır. (turn_camera_on işlevinin zaten tanımlandığını varsaymalısınız.)

İlk denemeniz öyleydi:

```
if (light < 0.01) or (temperature > 0.0):
    if not ((light < 0.01) and (temperature > 0.0)):
      turn_camera_on()
```

Bir arkadaşınız bunun özel bir veya (XOR) olduğunu ve daha basit bir şekilde şöyle yazılabileceğini söyledi:

```
if (light < 0.01) != (temperature > 0.0):
    turn_camera_on()
```

Arkadaşınız doğru mu? Eğer öyleyse, nedenini açıklayın. Eğer değilse, iki kod parçasının farklı sonuçlar üreteceği light ve temperature için değerler verin.

5. Python'un Sağladığı Fonksiyonlar bölümünde, sayfa 31'de, yerleşik işlev abs'i gördük. x değişkeni bir sayıyı referans verir. x ve mutlak değeri eşit olduğunda True döndüren ve aksi takdirde False döndüren bir ifade yazın. Sonuç değerini result adlı bir değişkene atayın.

6. different adında iki parametreye sahip bir işlev yazın: a ve b. parametreleri , a ve b farklı değerleri veriyorsa True, aksi takdirde False döndürsün.

7. Değişkenler population ve land_area ondalıklı sayılara (float) işaret ediyor.

   a. Nüfus 10.000.000'dan azsa nüfusu yazdıracak bir if ifadesi yazın.

   b. Nüfus 10.000.000 ile 35.000.000 arasında ise nüfusu yazdıracak bir if ifadesi yazın.

   c. Arazi yoğunluğu 100'den büyükse "Yoğun nüfuslu" yazdıracak if ifadesi yazın.

   d. Arazi yoğunluğu 100'den büyükse “Yoğun nüfuslu”, aksi takdirde “Seyrek nüfuslu” yazdıracak bir if ifadesi yazın.

8. Sayfa 35'teki Kendi Fonksiyonlarımızı Tanımlamak bölümündeki convert_to_celsius fonksiyonu, Fahrenheit'tan Celsius'a çeviriyor. Ancak, Wikipedia sekiz sıcaklık ölçeğini tartışıyor: Kelvin, Celsius, Fahrenheit, Rankine, Delisle, Newton, Rèaumur ve Rømer. Bu ölçeklerle ilgili daha fazla bilgi almak için Wikipedia'nın sıcaklık ölçekleri karşılaştırma sayfasına göz atın.

   a. convert_temperatures(t, source, target) fonksiyonunu yazın. Bu fonksiyon, sıcaklık t değerini kaynak biriminden (source) hedef birimine (target) çevirecek. source ve target her biri şu birimlerden biri olabilir: "Kelvin", "Celsius", "Fahrenheit", "Rankine", "Delisle", "Newton", "Reaumur" ve "Romer".

İpucu: Wikipedia sayfasında sekiz tablo var, her biri iki sütun ve yedi satır içeriyor. Bu, en az 8 * 7 if cümlesi anlamna gelir, çünkü sekiz birimin her biri dier yedi birime çevrilebilir. Muhtemelen daha da kötü, eğer baka bir scaklk ölçe ği eklemeye karar verirseniz, en az on alt yeni if cümlesi eklemeniz gerekir: sekiz tanesi yeni ölçeinizden mevcut birimlere ve sekiz tanesi de mevcut birimlerden yeni ölçeğe çeviri için.

Daha iyi bir yol, tek bir kanonik ölçek seçmektir, örnein Celsius. Dönüüm fonksiyonunuz iki admda çalabilir: önce kaynak ölçekten Celsius'a, ardndan Celsius'tan hedef ölçee dönütürme ilemi yaplabilir.

   b. Şimdi yeni bir sıcaklık ölçeği eklemek isterseniz, kaç if cümlesi eklemeniz gerekir?

9. Bir pH değeri 3.0'ın altındaysa güçlü bir uyarı mesajı yazdırmak ve aksi takdirde asitlik hakkında rapor vermek istiyoruz. Şu if ifadesini deniyoruz:

```
>>> ph = 2
>>> if ph < 7.0:
...     print(ph, "is acidic.")
... elif ph < 3.0:
...     print(ph, "is VERY acidic! Be careful.")
...
2 is acidic.
```

Bu kod, pH değeri 2 olduğunda yanlış mesaj yazdırıyor. Sorun nedir ve nasıl düzeltilir?

10. Aşağıdaki kod, bir çözeltinin asiditesi hakkında mesaj(lar) görüntüler:

```
ph = float(input("Enter the ph level: "))
if ph < 7.0:
    print("It's acidic!")
elif ph < 4.0:
    print("It's a strong acid!")
```

a. Kullanıcı 6.4 girdiğinde hangi mesaj(lar) görüntülenir?

b. Kullanıcı 3.6 girdiğinde hangi mesaj(lar) görüntülenir?

c. 4'ten küçük bir değer girildiğinde her iki mesajın da görüntülenmesi için koddaki bir satırı nasıl değiştirebilirsiniz?

11. Remembering Results of a Boolean Expression Evaluation bölümündeki son örnek neden bir kişinin hafif olup olmadığını (yani, BMI'sinin eşik değerden düşük olup olmadığını) kontrol ediyor, ağır olup olmadığını değil? Eğer ikinci atama ifadesini heavy = bmi >= 22.0 eklinde yazmak isterseniz, koda hangi değişiklik(ler)i yapmanız gerekir?