MERHABA PYTHON

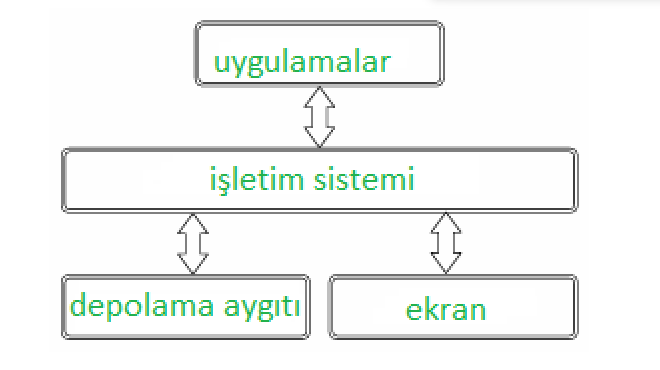
Programlar, bilgisayara ne yapacağını söyleyen komutlardan oluşur. Bu komutlara, bilgisayarın yürüttüğü ifadeler denir. Bu bölüm, Python'un en basit ifadelerini açıklar ve bunların bilgisayarlar için en yaygın görevlerden biri olan aritmetik işlemleri yapmayı nasıl kullanılabileceğini gösterir. Ayrıca programlamayı öğrenmeye başlamak için harika bir yerdir, diğer dillerinde temelidir.

BİLGİSAYARLAR PYTHON PROGRAMINI NASIL ÇALIŞTIRIR?

Programlama yaparken neyin ne olduğunu anlamak için, bir bilgisayarın bir programı nasıl yürüttüğüne dair zihinsel bir modele sahip olmak yardımcı olur.

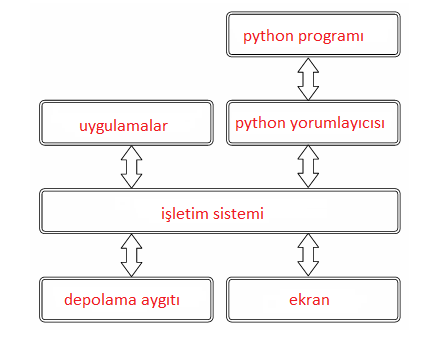
Bilgisayar, talimatları yürütebilen ve aritmetik yapabilen bir işlemci, sabit disk gibi verileri depolamak için bir yer ve ekran, klavye, ağa bağlanmak için bir Ethernet denetleyicisi gibi çeşitli diğer parçalar dahil olmak üzere donanım parçalarından oluşur.

Tüm bu parçalarla başa çıkmak için, her bilgisayar Microsoft Windows, Linux veya macOS gibi bir tür işletim sistemi çalıştırır. İşletim sistemi bir programdır; onu özel kılan şey, bilgisayarda donanıma doğrudan erişime izin verilen tek program olmasıdır. Başka herhangi bir uygulama (tarayıcınız, bir elektronik tablo programı veya bir oyun gibi) ekranda çizim yapmak, klavyede hangi tuşa basıldığını bulmak veya depodan veri almak istediğinde, işletim sistemine bir istek gönderir.



Bu, işleri yapmanın dolambaçlı bir yolu gibi görünebilir, ancak bu, yalnızca işletim sistemini yazan kişilerin bir grafik kartı ile diğeri arasındaki farklar ve bilgisayarın Ethernet veya kablosuz aracılığıyla bir ağa bağlı olup olmadığı konusunda endişelenmesi gerektiği anlamına gelir. Geri kalanımız—bilimsel verileri analiz eden veya 3 boyutlu sanal sohbet odaları oluşturan herkes—sadece işletim sistemini öğrenmek zorundadır ve programlarımız daha sonra binlerce farklı türde donanımda çalışacaktır.

Bugün, programcı ile bilgisayarın donanımı arasına başka bir katman eklemek yaygındır. Python, Java veya Visual Basic'te bir program yazdığınızda, doğrudan işletim sisteminin üzerinde çalışmaz. Bunun yerine, yorumlayıcı veya sanal makine adı verilen başka bir program programınızı alır ve sizin için çalıştırır, komutlarınızı işletim sisteminin anlayabileceği bir dile çevirir. İşletim sistemleri arasında doğrudan işletim sisteminin üzerine program yazmaktan çok daha kolay, daha güvenli ve daha taşınabilirdir:



Python yorumlayıcısını kullanmanın iki yolu vardır. Birincisi, .py uzantılı bir dosyada kayıtlı bir Python programını çalıştırmasını söylemektir. Diğeri ise, ifadeleri tek tek yazdığınız Shell adlı programda onunla etkileşim kurmaktır. Yorumlayıcı, yazdığınız her ifadeyi çalıştıracak, ifadenin yapmasını söylediği şeyi yapacak ve herhangi bir çıktıyı tek bir pencerede metin olarak gösterecek.

Python'u Şimdi Kurun (Daha Önce Kurmadıysanız) Python 3.6'yı henüz kurmadıysanız lütfen şimdi kurun. (Python 2 yeterli olmayacaktır; Python 2 ile Python 3 arasında önemli farklar vardır ve bu kitap Python 3.6'yı kullanır.) Kitabın web sitesinde kurulum talimatlarını bulun: http://pragprog.com/titles/gwpy3/ pratic-programming. Programlama pratik gerektirir: sadece bu kitabı okuyarak programlamayı öğrenemezsiniz, tıpkı gitar çalmayı anlatan bir kitap okuyarak gitar çalmayı öğrenemeyeceğiniz gibi. Python, Python programları yazmak için kullandığımız IDLE adlı bir programla birlikte gelir. IDLE, Python yorumlayıcısıyla iletişim kuran ve ayrıca bir dosyaya kaydedilen programları yazmanıza ve çalıştırmanıza olanak tanıyan bir Python kabuğuna sahiptir. IDLE'ı açmanızı ve örneklerimizi takip etmenizi şiddetle öneririz. Bu kitaptaki kodu yazmak, programlamada yeni bir dili öğrenirken eğitmeninize tekrarlanan ifadelere eşdeğerdir

İfadeler ve Değerler: Python'da Aritmetik

3 + 4 ("üç artı dört") ve 2- 3 / 5 ("iki eksi üç bölü beş") gibi matematiksel ifadeleri biliyoruz; her ifade 2, 3 ve 5 gibi değerlerden ve işlenenlerini farklı şekillerde birleştiren + ve - gibi operatörlerden oluşur. 4 / 5 ifadesinde operatör "/" ve işlenenler 4 ve 5'tir.

İfadelerin bir işleç içermesi gerekmez: bir sayı kendi başına bir ifadedir. Örneğin, 212'yi bir değeri olduğu kadar bir ifade olarak da kabul ederiz.

Herhangi bir programlama dili gibi, Python da temel matematiksel ifadeleri değerlendirebilir. Örneğin, aşağıdaki ifade 4 ve 13'ü toplar:

>>> 4 + 13

17

>>> sembolüne istem adı verilir. IDLE'ı açtığımızda, bu sembolün gösterildiği bir pencere açılmış olmalı; biz onu yazmayız. Bizi bir şey yazmaya yönlendirir. Burada 4 + 13 yazdık ve ardından bu ifadeyi girmeyi bitirdiğimizi belirtmek için Return (veya Enter) tuşuna bastık. Python daha sonra ifadeyi değerlendirdi.

Bir ifade değerlendirildiğinde, tek bir değer üretir. Önceki ifadede, 4 + 13'ün değerlendirilmesi 17 değerini üretti. İfadeyi Shell e yazdığımızda, Python üretilen değeri gösterir.

Çıkarma ve çarpma da aynı şekilde:

>>> 15 - 3

12

>>> 4 \* 7

28

Aşağıdaki ifadede 5'i 2'ye böler:

>>> 5 / 2

2.5

Sonuç bir ondalık noktaya sahiptir. Aslında, bölmenin sonucu her zaman bir ondalık noktaya sahiptir, sonuç tam sayı olsa bile:

>>> 4 / 2

2.0

Türler

Python'daki her değerin belirli bir türü vardır ve değer türleri, birleştirildiklerinde nasıl davranacaklarını belirler. 4 ve 17 gibi değerler int (tam sayının kısaltması) türüne sahiptir ve 2.5 ve 17.0 gibi değerler float türüne sahiptir. Float kelimesi, sayının basamakları arasında hareket eden ondalık noktayı ifade eden kayan noktanın kısaltmasıdır.

Aynı zamanda iki float içeren bir ifade bir float üretir:

>>> 17.0 - 10.0

7.0

Bir ifadenin işlenenleri bir int ve bir float olduğunda, Python otomatik olarak int'i bir float'a dönüştürür. Bu nedenle aşağıdaki iki ifade de aynı cevabı döndürür:

>>> 17.0 - 10

7.0

>>> 17 - 10.0

7.0

İstersek, kayan noktalı bir sayı yazarken ondalık noktadan sonraki sıfırı atlayabiliriz:

>>> 17 - 10.

7.0

>>> 17. - 10

7.0

Ancak çoğu kişi bunun kötü bir stil olduğunu düşünüyor çünkü programlarınızın okunmasını zorlaştırıyor: Ekranda bir noktayı kaçırmak ve 17. yerine 17 görmek daha kolay.

Tam Sayı Bölme, Modül ve Üs Alma

Arada sırada, bir bölme sonucunun yalnızca tam sayı kısmını isteriz. Örneğin, 53 saatte kaç tane 24 saatlik gün olduğunu bilmek isteyebiliriz. Gün sayısını hesaplamak için tam sayı bölmeyi kullanabiliriz:

>>> 53 // 24

2

Modül operatörünü kullanarak kaç saatin kaldığını bulabiliriz, bu da bölümün kalanını verir:

>>> 53 % 24

5

Python tam sayı bölümünün sonucunu yuvarlamaz. Bunun yerine, bölmenin sonucunun tabanını alır, bu da en yakın tam sayıya yuvarladığı anlamına gelir:

>>> 17 // 10

1

Negatif işlenenlerle % ve // ​​kullanırken dikkatli olmalıyız. Çünkü Python bir tamsayı bölümünün sonucunun tabanını alır, sonuç negatifse beklediğimizden bir küçüktür:

>>> -17 // 10

-2

Modül kullanıldığında, sonucun işareti bölenin (ikinci işlenen) işaretiyle eşleşir:

>>> -17 % 10

3

>>> 17 % -10

-3

Matematiksel olarak yatkın olanlar için, // ile % arasındaki ilişki şu denklemden gelir: sıfırdan farklı herhangi iki sayı a ve b için:

(b \* (a // b) + a % b) eşittir a

Örneğin, -17 // 10, -2'dir ve -17 % 10, 3'tür; o zaman 10 \* (-17 // 10) +-17 % 10, 10 \*-2 + 3 ile aynıdır, yani -17'dir.

Kayan nokta sayıları // ve % için de işlenenler olabilir. // ile bölme yapılır ve sonuç en yakın tam sayıya yuvarlanır, ancak tür bir kayan nokta sayısıdır:

>>> 3.3 // 1

3.0

>>> 3 // 1.0

3.0

>>> 3 // 1.1

2.0

>>> 3.5 // 1.1

3.0

>>> 3.5 // 1.3

2.0

Aşağıdaki ifade 3'ün 6. kuvvetini hesaplar:

>>> 3 \*\* 6

729

İki işleneni olan operatörlere ikili operatörler denir. Olumsuzluk, tek bir işlenene uygulandığı için tekli bir operatördür:

>>> -5

-5

>>> - -5

5

>>> - - -5

-5

Tip Nedir?

Artık iki tür sayı gördük (tamsayılar ve kayan nokta sayıları), bu yüzden bir türle neyi kastettiğimizi açıklamalıyız. Python'da bir tür iki şeyden oluşur:

• Bir değerler kümesi

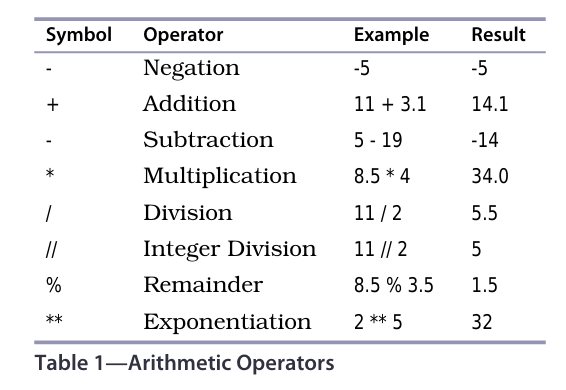
• Bu değerlere uygulanabilen bir işlemler kümesi

Örneğin, int türünde değerler …, -3, -2, -1, 0, 1, 2, 3, …'dir ve bu operatörlerin şu değerlere uygulanabileceğini gördük: +, -, \*, /, //, % ve \*\*.

Float türündeki değerler gerçek sayıların bir alt kümesidir ve aynı işlem kümesinin float değerlerine uygulanabilmesi mümkündür. Bunların çeşitli değerlere uygulandığında ne olduğunu, 13. sayfadaki Aritmetik İşleçler Tablo 1'de görebiliriz. Bir işleç birden fazla değer türüne uygulanabiliyorsa, buna aşırı yüklenmiş işleç denir.

Sonlu Kesinlik

Kayan nokta sayıları tam olarak ilkokulda öğrendiğiniz kesirler değildir. Örneğin, Python'un 2⁄3 ve 5⁄3 kesirlerinin versiyonuna bakın:



>>> 2 / 3

0.6666666666666666

>>> 5 / 3

1.6666666666666667

İlk değer 6 ile, ikincisi ise 7 ile biter. Bu şüphelidir: her ikisinin de ondalık noktadan sonra sonsuz sayıda 6'sı olmalıdır. Sorun, bilgisayarların sınırlı miktarda belleğe sahip olması ve (hesaplamaları hızlı ve belleği verimli hale getirmek için) çoğu programlama dilinin herhangi bir tek sayı için ne kadar bilgi depolanabileceğini sınırlamasıdır. 0,66666666666666666 sayısı, bilgisayarın bu sınırlı miktardaki bellekte gerçekten depolayabileceği 2⁄3'e en yakın değerdir ve 1,666666666666667, 5⁄3'ün gerçek değerine en yakın değerdir.

Operatör Önceliği

Fahrenheit'ı Celsius'a dönüştürmede int ve float bilgimizi kullanalım. Bunu yapmak için, Fahrenheit cinsinden sıcaklıktan 32'yi çıkaralım ve sonra 5⁄9 ile çarpalım:

>>> 212- 32 \* 5 / 9

194.22222222222223

Python, sonucun 194.222222222222223 santigrat derece olduğunu iddia ediyor, oysa aslında 100 olması gerekir. Sorun, çarpma ve bölmenin çıkarmadan daha yüksek önceliğe sahip olmasıdır; başka bir deyişle, bir ifade bir operatör karışımı içerdiğinde, \* ve / - ve +'dan önce değerlendirilir. Bu, aslında hesapladığımız şeyin 212- ((32 \* 5) / 9) olduğu anlamına gelir: 32 \* 5 alt ifadesi, bölme uygulanmadan önce değerlendirilir ve bu bölme, çıkarma gerçekleşmeden önce değerlendirilir.

Sayısal Kesinlik Hakkında Daha Fazlası Python'da tam sayılar (int türündeki değerler) istediğiniz kadar büyük veya küçük olabilir. Ancak, kayan noktalı sayılar yalnızca gerçek sayılara yaklaşık değerlerdir. Örneğin, 1⁄4 tam olarak saklanabilir ancak daha önce gördüğümüz gibi 2⁄3 saklanamaz. Daha fazla bellek kullanmak sorunu çözmez, ancak yaklaşık değeri gerçek değere daha yakın hale getirir, tıpkı 0.666'da 0'dan sonra daha fazla 6 yazmak onu tam olarak 2⁄3'e eşit yapmaz. 2⁄3 ile 0.666666666666666 arasındaki fark çok küçük görünebilir. Ancak bir hesaplamada 0.6666666666666666 kullanırsak, hata bileşik hale gelebilir. Örneğin, 1'i 2⁄3'e eklersek, ortaya çıkan değer ...6665 ile biter, bu nedenle birçok programlama dilinde 1 + 2⁄3, 5⁄3'e eşit değildir: >>> 2 / 3 + 1 1.6666666666666665 >>> 5 / 3 1.666666666666667 Daha fazla hesaplama yaptıkça, yuvarlama hataları giderek büyüyebilir, özellikle çok büyük ve çok küçük sayıları karıştırıyorsak. Örneğin, 10000000000 (10 milyar) ve 0,00000000001 (ondalık noktadan sonra 10 sıfır vardır) eklediğimizi varsayalım: >>> 10000000000 + 0,00000000001 10000000000.0 Sonuç, ilk ve son önemli basamak arasında yirmi sıfır içermelidir, ancak bu bilgisayarın depolayabileceğinden çok fazladır, bu nedenle sonuç yalnızca 10000000000 olur—sanki toplama hiç gerçekleşmemiş gibi. Büyük bir sayıya çok sayıda küçük sayı eklemenin hiçbir etkisi olmayabilir, bu da bir bankanın müşterilerinin tasarruf hesaplarının değerlerini topladığında istediği şey değildir. Kayan nokta sorununun farkında olmak önemlidir. Bunu çözmek için sihirli bir formül yoktur, çünkü bilgisayarlar hem bellek hem de hız açısından sınırlıdır. Sayısal analiz, sürekli matematiği yaklaşık olarak hesaplamak için algoritmaların incelenmesi, bilgisayar bilimi ve matematiğin en büyük alt alanlarından biridir. İşte bir ipucu: Kayan nokta sayılarını toplamanız gerekiyorsa, hatayı en aza indirmek için bunları en küçükten en büyüğe doğru toplayın.

Alt ifadelerin etrafına parantez koyarak öncelik sırasını değiştirebiliriz:

>>> (212- 32) \* 5 / 9

100.0

Tablo 2, En Yüksekten En Düşüğe Önceliğe Göre Listelenen Aritmetik Operatörler, sayfa 15'te aritmetik operatörler için öncelik sırasını göstermektedir.

Daha yüksek önceliğe sahip operatörler, daha düşük önceliğe sahip operatörlerden önce uygulanır. Bunu gösteren bir örnek şöyledir:

>>> -2 \*\* 4

-16

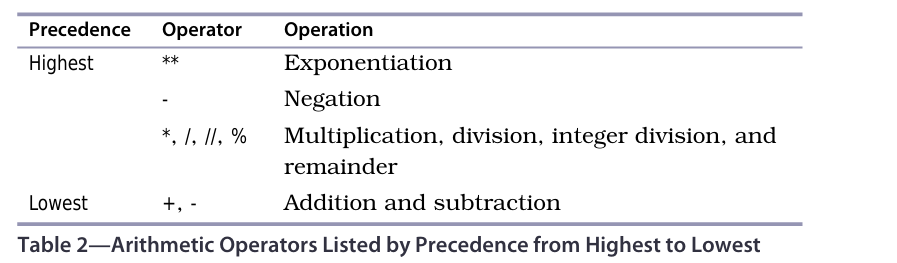
>>> -(2 \*\* 4)

-16

>>> (-2) \*\* 4

16

Üs alma, olumsuzlamadan daha yüksek önceliğe sahip olduğundan, 2 \*\* 4 alt ifadesi olumsuzlama uygulanmadan önce değerlendirilir.



Aynı satırdaki operatörler eşit önceliğe sahiptir ve üs alma işlemi hariç soldan sağa uygulanır, üs alma işlemi sağdan sola uygulanır. Yani, örneğin, ikili operatörler + ve - aynı satırda olduğundan, 3 + 4- 5, (3 + 4)- 5'e eşdeğerdir ve 3- 4 + 5, (3- 4) + 5'e eşdeğerdir.

Karmaşık ifadeleri parantez içine almak, 1 + 1.7 + 3.2 \* 4.4- 16 / 3 gibi şeyleri gözün okumasına yardımcı olduğu için, gerekmediğinde bile parantez içine almak iyi bir kuraldır. Öte yandan, 3.1 \* 5 gibi basit ifadelerde parantez kullanmamak iyi bir kuraldır.

Değişkenler ve Bilgisayar Belleği: Değerleri Hatırlama

Matematikçiler gibi, programcılar da sıklıkla değerleri daha sonra kullanabilmek için isimlendirirler. Bir değere atıfta bulunan bir isme değişken denir. Python'da, değişken isimleri harfler, rakamlar ve alt çizgi sembolü kullanabilir (ancak bir rakamla başlayamazlar). Örneğin, X, species5618 ve degrees\_celsius hepsine izin verilir, ancak 777'ye izin verilmez (bir sayıyla karıştırılır) ve hiçbiri de izin verilmez no-way! (noktalama işareti içerir). Değişken isimleri büyük/küçük harfe duyarlıdır, bu nedenle ph ve pH iki farklı isimdir.

Yeni bir değişkeni, ona bir değer atayarak yaratırsınız:

>>> degrees\_celsius = 26.0

Bu ifadeye atama ifadesi denir; degrees\_celsius'a 26.0 değerinin atandığını söyleriz. Bu degrees\_celsius'un 26.0 değerine atıfta bulunmasını sağlar. Değerleri kullanabildiğimiz her yerde değişkenleri kullanabiliriz. Python bir ifadede bir değişken gördüğünde, değişkenin atıfta bulunduğu değeri ikame eder:

>>> degrees\_celsius = 26.0

>>> degrees\_celsius

26.0

>>> 9 / 5 \* degrees\_celsius + 32

78.80000000000001

>>> degrees\_celsius / degrees\_celsius

1.0

Değişkenler, program yürütülürken değerleri değişebildiği için değişken olarak adlandırılır. Bir değişkene yeni bir değer atayabiliriz:

>>> degrees\_celsius = 26.0

>>> 9 / 5 \* degrees\_celsius + 32

78.80000000000001

>>> degrees\_celsius = 0.0

>>> 9 / 5 \* degrees\_celsius + 32

32.0

Zaten var olan bir değişkene bir değer atamak ikinci bir değişken yaratmaz. Bunun yerine, var olan değişken yeniden kullanılır, bu da değişkenin artık eski değerine başvurmadığı anlamına gelir.

Başka değişkenler de yaratabiliriz; bu örnek, suyun kaynama noktası ile santigrat derece olarak depolanan sıcaklık arasındaki farkı hesaplar:

>>> degrees\_celsius = 15.5

>>> difference = 100- degrees\_celsius

>>> difference

84.5

Uyarı: = Python'da Eşitlik Değildir! Matematikte = "soldaki şey sağdaki şeye eşittir" anlamına gelir. Python'da ise oldukça farklı bir şey ifade eder. Atama simetrik değildir: x = 12 x değişkenine 12 değerini atar, ancak 12 = x bir hatayla sonuçlanır. Bu nedenle, x = 12 ifadesini asla "x 12'ye eşittir" şeklinde tanımlamayız. Bunun yerine, bunu "x 12 alır" veya "x'e 12 atanır" şeklinde okuruz.