**Değerler, Değişkenler ve Bilgisayar Belleği**

Bir Python programı çalıştırıldığında, Python'un ne yaptığını takip edebilmemizi sağlayacak bir bilgisayar belleği modeli geliştireceğiz. Bu bellek modeli, Python kodu çalıştırıldığında ne olduğunu doğru bir şekilde tahmin edip açıklamamıza yardımcı olacak, ve bu beceri, iyi bir programcı olmak için gereklidir.

-----------------------------------------------------------------------------------------------------------

**Online Python Eğitmeni**  
Philip Guo, bellek modelimize oldukça iyi uyan, web tabanlı bir bellek görselleştirici yazdı. İşte URL: <http://pythontutor.com/visualize.html>. Hem Python 2 hem de Python 3 kodlarını izleyebilir; doğru sürümü seçtiğinizden emin olun. Bellek modelimize en yakın ayarlar şunlardır:  
• Çıkılmış çerçeveleri gizle  
• Tüm nesneleri heap üzerinde göster  
• İşaretçiler için metin etiketlerini kullan  
Python programının çalıştırılmasını izlemek istediğiniz her durumda bu görselleştiriciyi kullanmanızı şiddetle tavsiye ediyoruz.  
Eğer motive edici bulursanız, bellek modelimizi geliştirirken Philip'in görselleştiricisinden haberdar değildik (ve o da bizim modelimizden haberdar değildi), ancak yine de birbirlerine son derece yakındırlar.

--------------------------------------------------------------------------------------------------------------

Her bellek konumunun, tıpkı bir sokaktaki evin adresi gibi, o konumu benzersiz bir şekilde tanımlayan bir bellek adresi vardır.

Bellek adreslerimizi tamsayılardan farklı görünmeleri için **id** önekiyle işaretleyeceğiz(belirteceğiz.):

**id1**, **id2**, **id3** ve bu şekilde devam edecek.  
İşte float türündeki 26.0 değerini bellek modeliyle nasıl çizeceğimiz:

(Çizim burada .)



Bu görsel, id1 bellek adresindeki 26.0 değerini göstermektedir. Değerin türünü de her zaman göstereceğiz , bu örnekte değer türü float.

Bu kutuya bir **nesne** (object) diyeceğiz: bir bellek adresindeki bir türe sahip değer. Bir programın çalıştırılması sırasında, Python'un takip ettiği her değer, bilgisayar belleğindeki bir nesnenin içinde saklanır. Bellek modelimizde, bir değişken, referans verdiği objenin bellek adresini içerir:

metin, yazı tipi, beyaz, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

(**degrees\_celsius değişkeni id1 değerinde float değişken türüne sahip ve tuttuğu float değeri 26.0F dir.)**

Görselin yorumlanmasını kolaylaştırmak için, genellikle değişkenlerden nesnelerine doğru oklar çizeriz.

Aşağıdaki terminolojiyi kullanıyoruz:  
• 26.0 değeri **id1** bellek adresine sahiptir.  
• **id1** in bellek adresindeki nesnenin türü **float** ve değeri 26.0'dır.

• **degrees\_celsius** değişkeni, **id1** bellek adresini içerir.  
• **degrees\_celsius** değişkeni, 26.0 değerine referans verir.

Python, degrees\_celsius'in hangi değere referans verdiğini öğrenmesi gerektiğinde,degrees\_celsius in içerdiği bellek adresindeki nesneye bakar. Bu örnekte, o bellek adresi id1dir, bu yüzden Python id1 bellek adresindeki değeri kullanacaktır .Bu da 26.0 değeridir.

**Atama İfadesi**  
İşte bir atama ifadesinin genel formu:  
“değişken” = “ifade”  
Bu, şu şekilde gerçekleştirilir:

1. = işaretinin sağındaki ifadeyi değerlendirerek bir değer üretin. Bu değerin bir bellek adresi vardır.
2. Değerin bellek adresini = işaretinin solundaki değişkende tutun. Eğer o isim zaten mevcut değilse yeni bir değişken oluşturun eğer mevcutsa, mevcut değişkeni yeniden kullanarak içindeki bellek adresini değiştirin.

Bunu dikkate alalım :

metin, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

(burada float yani ondalıklı bir sayı ile bir tam sayı toplanlıp ondalıklı sayı türünde 31.0 float olarak yeni değer hesaplanmıştır .)

Python'un **degrees\_celsius = 26.0 + 5** değişkenini nasıl hesapladığı :

1. = işaretinin sağındaki ifadeyi hesaplayın : 26.0 + 5. Bu, bir bellek adresine sahip olan 31.0 değerini ortaya çıkarır . (Unutmayın ki Python tüm değerleri bilgisayar belleğinde saklar.)
2. = işaretinin solundaki değişken olan **degrees\_celsius**'un, 31.0 değerine referans vermesi için, 31.0 ın bellek adresini **degrees\_celsius** da tutun.

Değişkenlere Yeniden Atama (değerini değiştirme)

Burayı Dikkate Alalım:

metin, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu kod, bir değişkene atama yapmanın başka herhangi bir değişkeni değiştirmediğini gösterir. **difference** değişkenine 20 değeri atayarak başlıyoruz, ardından 2 \* **difference** ifadesinin sonucunu (bu 40'ı üretir) **double** değişkenine atıyoruz.

Sonraki adımda, **difference** değişkenine 5 değeri atıyoruz; ancak **double** değişkeninin değerine baktığımızda, hâlâ 40 ı referans verdiğini görüyoruz.

Bu, kurallarımıza göre şöyle çalışır. İlk ifade, **difference = 20**, şu şekilde yürütülür:

1. = işaretinin sağındaki ifadeyi değerlendirin: 20. Bu, bellek adresi **id1**'de saklayacağımız 20 değerini üretir.
2. = işaretinin solundaki değişken olan **difference**'in, 20'ye referans vermesi için **id1**'i **difference**'da saklayın.

Bellek modelinin mevcut durumu budur. (**double** değişkeni henüz oluşturulmamıştır çünkü ona atama işlemini henüz gerçekleştirmedik.)

metin, yazı tipi, beyaz, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

İkinci ifade, **double = 2 \* difference**, şu şekilde yürütülür:

1. = işaretinin sağındaki ifadeyi değerlendirin: 2 \* **difference**. Bellek modelinde **difference**'in 20 değerine referans verdiğini gördüğümüz için, bu ifade 2 \* 20 ile eşdeğerdir ve 40 değerini üretir. 40 için bellek adresi **id2**'yi seçeceğiz
2. = işaretinin solundaki değişken olan **double**'un, 40'a referans vermesi için **id2**'yi **double**'da saklayın.

Bellek modelinin şuanki durumu budur:

metin, yazı tipi, ekran görüntüsü, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Python, üçüncü ifade olan **double**'ı yürüttüğünde, yalnızca **double**'ın referans verdiği değeri (40) arar ve gösterir.

Dördüncü ifade, **difference = 5**, şu şekilde yürütülür:

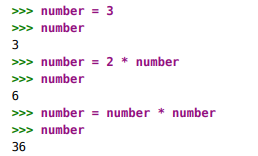
1. = işaretinin sağındaki ifadeyi değerlendirin: 5. Bu, bellek adresi **id3**'de saklayacağımız 5 değerini üretir.
2. = işaretinin solundaki değişken olan **difference**'in, 5'e referans vermesi için **id3**'yi **difference**'da saklayın.

Bellek modelinin şuanki durumu budur:

metin, diyagram, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**double** değişkeni hâlâ **id2**'yi içerdiği için, hâlâ 40'a referans vermektedir. Artık hiçbir değişken 20'ye referans vermiyor.  
Beşinci ve son ifade olan **double**, yalnızca **double**'ın referans verdiği değeri, yani hâlâ 40'ı arar ve gösterir.  
Bir atama ifadesinin her iki tarafında da bir değişken kullanabiliriz:



Artık Python'un bu kodu nasıl yürüttüğünü açıklayacağız, ancak bellek adreslerini açıkça belirtmeyeceğiz. Ne olduğunu tarif ederken bunu bir kağıda izleyin; bu sırada kendi bellek adreslerinizi oluşturun.  
Python, ilk ifadeyi **number = 3** şu şekilde yürütür:

1. = işaretinin sağındaki ifadeyi değerlendirin: 3. Bu ifadeyi değerlendirmek kolaydır: 3 üretilir.
2. = işaretinin solundaki değişken olan **number**'ın, 3'e referans vermesini sağlayın.

Python, ikinci ifadeyi **number = 2 \* number** şu şekilde yürütür:

1. = işaretinin sağındaki ifadeyi değerlendirin: 2 \* **number**. **number** şu anda 3'e referans verdiği için, bu 2 \* 3 ile eşdeğerdir ve 6 üretilir.
2. = işaretinin solundaki değişken olan **number**'ın, 6'ya referans vermesini sağlayın.

Python, üçüncü ifadeyi **number = number \* number** şu şekilde yürütür:

1. = işaretinin sağındaki ifadeyi değerlendirin: **number** \* **number**. **number** şu anda 6'ya referans verdiği için, bu 6 \* 6 ile eşdeğerdir ve 36 üretilir.
2. = işaretinin solundaki değişken olan **number**'ın, 36'ya referans vermesini sağlayın.

Arttırılmış Atama

Bu örnekte, **score** değişkeni atama ifadesinin her iki tarafında da yer alır:

İfade :

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu o kadar yaygındır ki, Python bu işlem için bir kısayol notasyonu sağlar:

Notasyon :

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Arttırılmış bir atama, bir atama ifadesini bir operatörle birleştirerek ifadeyi daha özlü hale getirir. Bir artırılmış atama ifadesi şu şekilde yürütülür:

1. = işaretinin sağındaki ifadeyi değerlendirerek bir değer üretin.
2. = işaretine ekli operatörü, = işaretinin solundaki değişken ile üretilen değer üzerinde uygulayın. Bu, başka bir değer üretir. O değerin bellek adresini = işaretinin solundaki değişkende saklayın.  
   Not edin ki, operatör, sağdaki ifade değerlendirildikten sonra uygulanır:

yazı tipi, beyaz, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tablo 2, Sayısal Operatörlerin Öncelik Sırasına Göre Listesi (En Yüksekten En Düşüğe) sayfa 15'te yer alan tüm operatörlerin (negasyon hariç) kısayol sürümleri vardır.

Örneğin:

bir sayının karesini almak için onu kendisiyle çarparak yapabiliriz:

metin, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu kod, şuna denktir:

metin, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

(ikiside aynı sonucu verir fakat ilk resimdeki kısayoldur ve daha çok karşımıza çıkar)

Tablo 3

gördüğünüz artırılmış operatörlerin bir özetini ve **İfadeler ve Değerler: Python'da Aritmetik** sayfa 9'da öğrendiğiniz sayısal operatörlere dayanan birkaç tane daha içerir.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**(Tablo 3—Artırılmış Atama Operatörleri)**

**Python Size Bir Şeylerin Yanlış Gittiğini Nasıl Bildirir**

**Genel anlamda, Python'da iki tür hata vardır:**

**Yazım hataları: geçerli bir Python kodu yazmadığınızda meydana gelir.**

**Anlamsal hatalar : Python'a yapamayacağı bir şeyi yaptırmaya çalıştığınızda, örneğin bir sayıyı sıfıra bölmek veya var olmayan bir değişkeni kullanmaya çalışmak gibi durumlarda ortaya çıkar.**

Henüz oluşturulmamış bir değişkeni kullanmaya çalıştığımızda olanlar şunlardır:

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

(moogah tanımlı değil veya bulunamıyor tarzında bir hata mesajı vermekte )

Bu oldukça karmaşık;

Python hata mesajları, zaten Python bilen insanlar için tasarlanmıştır. (Onlara alışacaksınız ve yararlı bulacaksınız.) Ancak, ilk iki satır şu anda pek yararlı değil, ama daha uzun programlar yazmaya başladığımızda vazgeçilmez olacaklar.

Son satır ise neyin yanlış gittiğini belirtiyor: moogah adı tanınmadı.

İşte bazen görebileceğiniz başka bir hata mesajı:

metin, yazı tipi, beyaz, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

(kod hatası : kod eksik veya yanlış yazılınca ortaya çıkar)

Bir programlama dilinde neyin serbest (izin verilen durum) olduğunu belirleyen kurallara **sözdizimi (syntax)** denir. Mesajda Python'un sözdizimi kurallarını ihlal ettiğimizi bildiriyor. Bu durumda, 2 ye bir şey eklemesini istemişiz fakat ne ekleyeceğimizi belirtmemişiz.

Daha önce 16.sayfada Python da 12 = x ifadesinin hatalı bir ifade olduğunu belirtmiştik. Haydi Deneyelim.

(Pythonda = işareti eşitlik değildir!)



(Kod Hatası : Değişken Bir sabite atanamaz!)

Bir sabit 12 ve 26.0 gibi rastgele bir değerdir.

Bu bir “**SyntaxError”** dur çünkü: Python o atama ifadesini incelerken bir sayıya değer atayamayacağınızı bilir bu yüzden 12'nin değerini başka bir şeye yapamazsınız.12 sadece 12 dir.