**Python Programlama 101**  **1**

Bu Bilgisayar Bilimleri metni, ilk CS metninizde veya kursunuzda öğrendiğiniz becerileri daha da geliştirir ve büyük miktarda veriyle başa çıkmak için verimli algoritmaları nasıl kullanacağınızı öğreterek çantanıza ekler. Doğru verimlilik anlayışı olmadan, büyük veri setleriyle çalışırken en hızlı bilgisayarları bile durma noktasına getirmek mümkündür. Bu daha önce de oldu ve yakında bunun ne kadar kolay gerçekleşebileceğini anlayacaksınız. Ama önce, programlama için bazı kalıpları gözden geçireceğiz ve dilin temel yapısını ve sözdizimini anladığınızdan emin olmak için Python programlama diline bakın.

Python kullanarak program yazmaya başlamak için Python'u bilgisayarınıza yüklemeniz gerekir. Bu metindeki örnekler Python 3 kullanmaktadır. Python 2, Python 3 ile uyumlu değildir, bu nedenle bilgisayarınızda Python 3 veya sonraki bir sürümün yüklü olduğundan emin olmak isteyeceksiniz. Herhangi bir dilde program yazarken iyi bir Entegre Geliştirme Ortamı (IDE) değerli bir araçtır, bu nedenle siz de bir IDE kurmak isteyeceksiniz. Bu metindeki örneklerde Şekil 1.1'de gösterildiği gibi Wing IDE 101 kullanılacaktır, ancak diğer kabul edilebilir IDE'ler de mevcuttur. The Wing IDE iyi korunur, kullanımı basittir ve Python programları yazarken faydalı olacak güzel bir hata ayıklayıcıya sahiptir. Wing IDE 101 almak istiyorsanız <http://wingware.com> adresine gidin. <http://cs.luther.edu/~leekent/CS1> web sitesinde hem Python 3 hem de Wing IDE 101'i kurmak için talimatlar bulunmaktadır. Wing IDE 101, eğitim amaçlı kullanım için Wing'in ücretsiz sürümüdür.

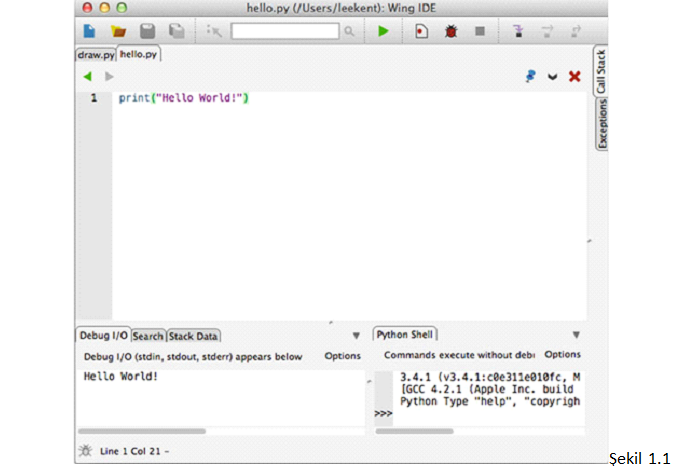
Python hakkında metni okurken bilmeniz gereken bazı genel kavramlar vardır. Python yorumlanmış bir dildir. Bu, Python kodunu yazdıktan sonra çalıştırmadan önce herhangi bir ekstra adımdan geçmeniz gerekmediği anlamına gelir. Wing IDE'deki hata ayıklama düğmesine basabilirsiniz (bir böceğe benziyor) ve daha önce en az bir kez yapmadıysanız programınızı kaydetmenizi isteyecektir. Daha sonra programınızı çalıştıracaktır. Python ayrıca dinamik olarak yazılır. Bu, bazı programlama dillerinde olduğu gibi programınızı çalıştırmadan önce herhangi bir tür hatası almayacağınız anlamına gelir. Programınızda kullandığınız veri türlerini anlamanız özellikle önemlidir. Bu konuda birazdan daha fazla bilgi vereceğim. Son olarak, Python programları Python yorumlayıcısı tarafından yorumlanır. Kabuk, Python yorumlayıcısının diğer bir adıdır ve Wing IDE 101, IDE'nin içinde bir kabuğa erişmenizi sağlar

© Springer International Publishing Switzerland 2015

K.D. Lee and S. Hubbard, Data Structures and Algorithms with Python,

Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-13072-9\_1

1 Python Programlama 101



Python deyimlerini ve ifadelerini Python Shell yazan pencere bölmesine yazarak bir programa koymadan önce bir kod parçasını hızlıca deneyebilirsiniz. Çoğu programlama dilinde olduğu gibi, Python programlarınızda alabileceğiniz birkaç tür hata vardır. Sözdizimi hataları programınız çalışmadan önce bulunur. Bunlar iki nokta üst üste işaretini atlamak veya bir şeyi girintilemeyi unutmak gibi şeylerdir. Wing IDE 101 gibi bir IDE bu sözdizimi hatalarını vurgulayacaktır, böylece bunları düzeltebilirsiniz. Çalışma zamanı hataları, programınız çalıştığında bulunur. Çalışma zamanı hataları, beklenmedik değerlere sahip değişkenler ve bu değerler üzerindeki işlemler gibi şeylerden kaynaklanır. Bir çalışma zamanı hatasını bulmak için Şekil 1.1'de göründüğü gibi Yığın Verisi sekmesine bakabilirsiniz. Bir çalışma zamanı hatası oluştuğunda program yürütülmeyi durdurur ve Yığın Verileri sekmesi program değişkenlerini görebileceğiniz çalışma zamanı yığınını incelemenizi sağlar.

Bir sorunu hala anlamamanız durumunda, Wing IDE 101 (ve diğer IDE'lerin çoğu) kodunuzda adım adım ilerlemenizi sağlar, böylece bir hatanın nasıl yeniden üretildiğini izleyebilirsiniz. Şekil 1.1'in sağ üst köşesindeki üç simge sırasıyla bir fonksiyona adım atmanızı, kodun üzerinden geçmenizi ve bir fonksiyondan çıkmanızı sağlar. Bir çalışma zamanı hatasını ve bunun nasıl oluştuğunu anlamaya çalışırken kodunuzun üzerinden veya içinden geçmek değerli olabilir.

Çok belirgin olmayan bir başka araç da Wing IDE tarafından sağlanmaktadır. IDE'nin sol tarafındaki satır numarasına tıklayarak bir kesme noktası ayarlamak mümkündür. Bir kesme noktası, programın kesme noktasından hemen önce yürütmeyi durdurmasına neden olur. Buradan, bir hatanın nasıl oluştuğunu belirlemek için kodunuz üzerinde adım atmaya başlamak mümkündür.

Bu bölümde Python öğrenmeye veya gözden geçirmeye motive etmek için, metin kaplumbağa grafikleri ve Tkinter adlı bir Grafik Kullanıcı Arayüzü (GUI) çerçevesi kullanarak basit bir çizim uygulaması geliştirecektir. Yol boyunca, akümülatör kalıbı ve bir dosyadan kayıt okumak için döngü ve yarım kalıbı dahil olmak üzere bazı programlama kalıplarını keşfedeceksiniz. Ayrıca Python'da

1 Python Programlama

fonksiyonları görecek ve bir sınıf tanımı tasarlayıp yazarak kendi veri türlerinizi nasıl uygulayacağınızı öğrenmeye başlayacaksınız.

**1.1 Bölüm Hedefleri**

Bu bölümün sonunda bu sorulara cevap verebiliyor olmalısınız.

* Accumulator kalıbı için hangi iki parça gereklidir?
* Bir dosyadan okumak için döngü ve yarım kalıbını ne zaman kullanmanız gerekir?
* Bir sınıf tanımının amacı nedir?
* Nesne nedir ve nasıl oluşturulur?
* Mutator yöntemi nedir?
* Accessor metodu nedir?
* Widget nedir ve GUI programlamada widget'lar nasıl kullanılır?

**1.2 Nesne Oluşturma**

Python nesne yönelimli bir dildir. Python'daki tüm veri öğeleri nesnelerdir. Python'da, benzer olarak düşünülebilecek veri öğeleri bir tür veya sınıf ile adlandırılır. Python'da tip ve sınıf terimleri eş anlamlıdır: aynı şey için iki isimdir. Yani Python'da tipleri okuduğunuzda sınıfları düşünebilirsiniz ya da tam tersi. Python'da int, float, str, list ve sözlüğün kısaltması olan dict dahil olmak üzere çeşitli yerleşik veri türleri vardır. Bu veri türleri ve bunlarla ilgili işlemler metnin sonundaki eklere dahil edilmiştir, böylece programlama sırasında başvurmanız gerektiğinde hızlı bir referansa sahip olursunuz. Ayrıca Python kabuğunda help(typename) yazarak herhangi bir tür için yardım alabilirsiniz, burada typename Python'da bir tür veya sınıftır. Çok iyi bir dil referansı, resmi Python dokümantasyon web sitesi olan <http://python.org/doc> adresinde bulunabilir.

**1.2.1 Literal Değerler**

Python'da nesne oluşturmanın iki yolu vardır. Birkaç durumda, bir nesne oluşturmak için değişmez bir değer kullanabilirsiniz. Literal değerler, programımız içinde bir değişkeni belirli bir değere ayarlamak istediğimizde kullanılır. Örneğin, 6 değişmezi, 6 tamsayı değerine sahip herhangi bir nesneyi belirtir.

X=6

Bu, 6 değerini içeren bir int nesnesi oluşturur. Ayrıca Şekil 1.2'de gösterildiği gibi x adlı referansı bu nesneye işaret eder. Python'daki tüm atamalar referansları nesnelere yönlendirir.

1 Python Programlama 101

  
Bir atama deyimi gördüğünüzde, eşittir işaretinin sol tarafındaki şeyin bir referans olduğunu ve sağ taraftaki şeyin ya başka bir referans ya da yeni oluşturulmuş bir nesne olduğunu hatırlamalısınız. Bu durumda, x = 6 yazmak yeni bir nesne oluşturur ve ardından x'i bu nesneye işaret eder.

Python'da başka literal değerler de yazılabilir. İşte Python'da mümkün olan bazı literal değerler.

* int değişmezleri: 6, 3, 10, -2, vb.
* float değişmezleri: 6.0, -3.2, 4.5E10
* str değişmezleri: 'merhaba', “nasılsın”
* liste değişmezleri: [], [6, 'merhaba']
* dict değişmezleri: {}, {'merhaba':6, 'nasılsın':4}

Python, float değişmezleri bir üs ile belirtmenize izin verir.

Yani, 4.5E10, 45000000000.0 float değerini temsil eder. Ondalık sayı ile yazılan herhangi bir sayı

noktasının bir float olduğunu, ondalık noktadan sonra 0 veya başka bir değer olup olmadığını belirtir. Eğer bir sayıyı E veya üslü gösterim kullanarak yazarsanız, bu da bir float olur. Herhangi bir sayı

E gösteriminde yazılmadığı sürece, ondalık noktası olmayan bir int değeridir. Dize değişmezleri şunlardır

tek veya çift tırnak ile çevrelenir. Liste değişmezleri [ ve ] ile çevrelenir. List [] değişmezi boş listeyi temsil eder. Dict {} değişmezi boş sözlüktür.

Daha önce sözlük kullanmamış olabilirsiniz. Sözlük, anahtarların bir eşlemesidir değerlerle eşleştirilir. Sözlük değişmezinde, 'merhaba' anahtarı 6 değeriyle eşlenir ve 'nasılsınız' anahtarı 4 ile eşleştirilmiştir. Sözlükler 5. Bölümde ayrıntılı olarak ele alınacaktır.

**1.2.2 Gerçek Olmayan Nesne Oluşturma**

Çoğu zaman, bir nesne oluşturulduğunda, gerçek bir değerden oluşturulmaz. Elbette programlama dillerinde değişmez değerlere ihtiyaç duyarız, ancak çoğu zaman zaten bir nesnemiz vardır ve mevcut bir veya daha fazla nesneyi kullanarak başka bir nesne oluşturmak isteriz. Örneğin Python'da elimizde '6' gibi bir karakter dizisi varsa ve bu karakter dizisinden bir int nesnesi oluşturmak istiyorsak şunları yapabiliriz.

y = ’6’

x = int(y)

print(x)

1 Python Programlama 101

Bu kısa kod parçasında y, string literalinden oluşturulan str nesnesine bir referanstır. x değişkeni, y'nin atıfta bulunduğu nesne kullanılarak oluşturulan bir nesneye referanstır. Genel olarak, başka nesne değerlerine dayalı bir nesne oluşturmak istediğimizde aşağıdakileri yazarız:

*variable = type(other\_object\_values)*

Tür, Python'da int, float, str veya başka bir tür gibi herhangi bir tür veya sınıf adıdır. other\_object\_values, bu türün bir örneğini (yani bir nesnesini) oluşturmak için sınıf veya tür tarafından ihtiyaç duyulan diğer nesnelere virgülle ayrılmış bir referans dizisidir. Aşağıda, gerçek olmayan değerlerden nesne oluşturmaya ilişkin bazı örnekler verilmiştir.

*z = float('6.3')*

*w = str(z)*

*u = list(w) # bu, ['6', '.', '3'] listesiyle sonuçlanır*

**1.3 Nesneler Üzerinde Yöntem Çağırma**

Nesneler kullanışlıdır çünkü ilgili bilgileri toplamamıza ve bu veriler üzerinde hareket eden davranışlarla gruplandırmamıza izin verirler. Bu davranışlara Python'da metot denir. Herhangi bir nesne yönelimli dilde iki tür yöntem vardır: mutator ve accessor yöntemleri. Erişimci yöntemler bir nesnenin geçerli durumuna erişir ancak nesneyi değiştirmez. Accessor yöntemleri çağrıldığında yeni nesne referansları döndürür.

*X= ‘how are you’*

*Y= x.upper()*

*print(y)*

Burada, upper yöntemi x'in başvurduğu nesne üzerinde çağrılır. upper accessor yöntemi, orijinal dizenin büyük harfli bir sürümü olan yeni bir nesne, bir str nesnesi döndürür. x üzerinde upper yönteminin çağrılmasıyla x'in değişmediğine dikkat edin. upper yöntemi bir accessor yöntemdir. Str türünde, eklerde öğrenebileceğiniz birçok accessor yöntem mevcuttur. Bazı yöntemler mutator yöntemlerdir. Bu metotlar aslında var olan nesneyi değiştirir. Bunun iyi bir örneği, liste türündeki tersine çevirme yöntemidir.

1 Python Programlama 101

*myList=[1,2,3]*

*MyList.reverse()*

*print(myList) # Bu ekrana [3,2,1] yazdırır.*

Ters yöntem, mevcut nesneyi, bu durumda myList'in başvurduğu listeyi mutasyona uğratır. Bir mutatör yöntemi çağrıldıktan sonra geri alınamaz. Değişiklik veya mutasyon, başka bir mutator yöntemi tarafından tekrar mutasyona uğratılana kadar kalıcıdır.

Tüm sınıflar accessor yöntemler içerir. Erişimci yöntemler olmasaydı, sınıf oldukça ilgisiz olurdu. Bir nesnede depolanan bir değeri almak veya bir nesnede depolanan değere bağlı olan bir değeri almak için erişimci yöntemleri kullanırız.

Eğer bir sınıfın accessor metotları olmasaydı, nesneye değerler koyabilirdik ama onları asla geri alamazdık Bazı sınıfların mutator metotları vardır ve bazılarının yoktur. Örneğin, liste sınıfının tersine çevirme yöntemi de dahil olmak üzere mutator yöntemleri vardır. Herhangi bir mutator yöntemi olmayan bazı sınıflar vardır. Örneğin, str sınıfının herhangi bir mutator yöntemi yoktur. Bir sınıf herhangi bir mutator metodu içermiyorsa, o sınıfın değişmez olduğunu söyleriz. Değişmez bir sınıftaki verilerden yeni değerler oluşturabiliriz, ancak değişmez bir nesne oluşturulduktan sonra değiştirilemez. Diğer değişmez sınıflar arasında int ve float bulunur.

**1.4 Bir Sınıfın Uygulanması**

Nesne yönelimli bir dilde programlama genellikle yazdığınız programın ihtiyaç duyduğu bilgileri tutan nesneleri tanımlayan sınıfları uygulamak anlamına gelir. Nesneler veri içerir ve yöntemler bu veriler üzerinde işlem yapar. Bir sınıf, belirli bir nesne türü için veri ve yöntemlerin tanımıdır.

Her sınıf, constituent adı verilen özel bir yöntem içerir. Constituent’in görevi, nesnenin içindeki verilere referanslar yerleştirerek bir nesnenin örneğini oluşturmaktır. Örneğin, Dog adlı bir sınıf düşünün. Bir köpeğin bir adı, bir doğum günü ve havladığında çıkardığı bir ses vardır. Bir Dog nesnesi oluşturduğumuzda, Bölüm 1.4.1'de görünen gibi bir kod yazarız.

1 Python Programlama 101

**1.4.1 Nesne Oluşturma ve Yöntem Çağırma**

*1 boyDog = Dog("Mesa", 5, 15, 2004, "WOOOF")*

*2 girlDog = Dog("Sequoia", 5, 6, 2004, "barkbark")*

*3 print(boyDog.speak())*

*4 print(girlDog.speak())*

*5 print(boyDog.birthDate())*  *6* *print(girlDog.birthDate())*

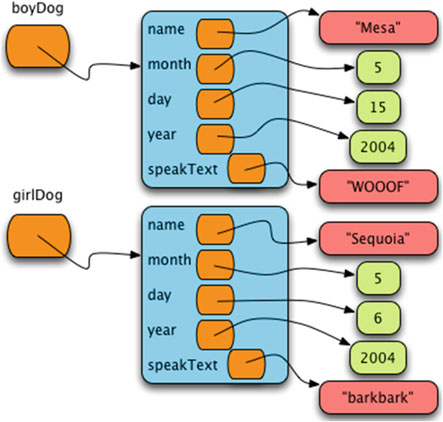
*7 boyDog.changeBark("woofywoofy")*

*8 print(boyDog.speak())*

Bilgisayarın belleğinde oluşturulduktan sonra, köpek nesneleri Şekil 1.3'te görünenlere benzer. Her nesne, kendisine atanan değişken referansı ile referanslandırılır; bu durumda girlDog ya da boyDog. Nesnelerin kendileri, nesnede depolanan bilgilere işaret eden bir referanslar koleksiyonudur. Her nesne, bir Dog nesnesini oluşturan ilişkili verilere işaret eden name, month, day, year ve speakText referanslarına sahiptir.

Bu iki nesne gibi Dog nesneleri oluşturabilmek için bu nesneleri tanımlayacak bir Dog sınıfına ihtiyacımız vardır. Buna ek olarak, speak, birthDate ve changeBark yöntemlerini tanımlamamız gerekecektir. 1.4.2. Bölümde gösterildiği gibi bir sınıf yazarak bunu yapabiliriz. Sınıfın her bir parçası ile ilgili açıklamalar kodda yer almaktadır. Özel değişken self her zaman geçerli nesneyi gösterir ve sınıftaki her yöntemin ilk parametresi olmalıdır.

1 Python Programlama 101



Şekil 1.3 Bir Çift Köpek Nesnesi

Python self argümanını metotlara aktarmakla ilgilenir. Diğer argümanlar, yöntem çağrıldığında programcı tarafından aktarılır (Bölüm 1.4.1'deki her bir yöntemin çağrılması örneğine bakın).

**1.4.2 Dog Sınıfı**

**1 class Dog:**

*2 # Bu, sınıf için kurucudur. Bir Dog nesnesi oluşturulduğunda çağrılır.*

1 Python Programlama 101

*3 # nesne oluşturulur. “self” adlı referans Python tarafından oluşturulur .*

*4 # ve yeni oluşturulan nesnenin alanına işaret etmesi sağlanır.*

*5 # Python bunu bizim için otomatik olarak yapar, ancak*

*6 #init metodunun (yani kurucunun) ilk parametresinin “self” olması gerekir.*

*7 def \_\_****init\_\_****(self, name, month, day, year, speakText):*

*8 self.name = name*

*9*  *self.month = month*

*10*  *self.day = day*

*11*  *self.year = year*

*12*  *self.speakText = speakText*

*13*

*14 # Bu, nesnede depolanan speakText'i döndüren bir erişimci*

*15 # yöntemidir. “self “in bir parametre olduğuna dikkat edin. Her yöntemin ilk parametresi*

*16 # “self”tir. “self” parametresi geçerli nesneye bir referanstır.*

*17 # Geçerli nesne, yöntem çağrıldığında noktanın (yani .) sol tarafında görünür.*

*19*  ***def*** *speak(self):*

1 Python Programlama 101

*20*  ***return*** *self.speakText*

*21*

# *22 # İşte adı almak için bir accessor yöntemi*

*23* ***def*** *getName(self):*

*24*  ***return*** *self.name*

# *25 #**Bu, tarihi temsil eden bir dize döndürmek için*

# *26 # doğum günü bilgilerini kullanan başka bir erişimci yöntemidir.*

*27*  ***def*** *birthDate(self):*

*29* ***return*** *str(self.month) + "/" + str(self.day) + "/" + str(self.year)*

*30*

# 31 # *Bu, Dog nesnesinin speakText'ini değiştiren bir mutator yöntemidir.*

*32*  ***def*** *changeBark(self,bark):*

*33*  *self.speakText = bark*

**1.5 Operatör Aşırı Yükleme**

Python, programlama dilinin güzel bir özelliği olan operatör aşırı yüklemesi sağlar çünkü programcının nesnelerle çok doğal bir şekilde etkileşime girmesini mümkün kılar. Operatör aşırı yüklemesi, çeşitli operatörler için zaten uygulanmaktadır. Örneğin Python'da yerleşik sınıflar veya türlerde. Örneğin, tamsayılar (yani int türü) yeni bir tamsayı nesnesi oluşturmak için nasıl bir araya getirilebileceği.

Python'da \_\_add\_\_ yöntemi adı verilen özel bir yöntemle. İki tamsayı bir araya getirildiğinde, bu yöntem yeni bir tamsayı nesnesi oluşturmak için çağrılır. Eğer bakarsanız eklerinde, bu özel yöntemlerin örneklerini ve nasıl çağrıldıklarını göreceksiniz. Örneğin, Bölüm.13'te \_\_add\_\_ yöntemi x + y yazılarak çağrılır ve burada x bir tam sayıdır. İki alt çizgi ile başlayan ve biten yöntemler, Python'un karşılık gelen bir operatörle ilişkilendirdiği yöntemlerdir. Python'un operatör aşırı yüklemesini desteklediğini söylediğimizde, eğer sınıfınız için operatör tarafından aşırı yüklenmiş bir ada sahip bir yöntem tanımlarsanız, sınıfınız bu operatörü de destekleyecektir. Python hangi metodun çağrılacağını ilgili işlenenlerin türlerine göre belirler. Örneğin, x + y yazmak int sınıfını çağırır x bir tamsayı olduğunda \_\_add\_\_ yöntemi, ancak float türünün \_\_add\_\_ yöntemini çağırır Bunun nedeni, \_\_add\_\_metodu söz konusu olduğunda, nesnenin operatörünün sol tarafı, + operatörünün sol tarafındaki nesneye karşılık gelir.x.\_\_add\_\_(y) eşdeğer yöntem çağrısında nokta (yani nokta). Soldaki nesne noktasının tarafı hangi add yönteminin çağrılacağını belirler. Operatör aşırı yüklenmiştir.Dog sınıfımız için toplama işlemini tanımlamak istersek, bir \_\_add\_\_ ErkekKöpek+KızKöpek yazarak Yeni bir dog nesnesi. Eğer bunu yapmak isteseydik, şu şekilde olurdu (Sect.1.5.1'de.)

**1.5.1 Aşırı Yüklü Eklentili Dog Sınıfı**

1 **sınıf Dog:**

2 # Bu, sınıf için kurucudur. Ne zaman bir dog

3 # nesne oluşturulur. “self” adlı referans Python tarafından oluşturulur

4 # ve yeni oluşturulan nesnenin alanına işaret etmesi sağlanır. Python

5 # bunu bizim için otomatik olarak yapar ancak ilk olarak "self "e sahip olmamız gerekir

6 # \_\_init\_\_ yöntemine (yani kurucuya) parametre

7 **def** \_\_init\_\_(self, name, month, day, year, speakText):

8 self.name = name

9 self.month = ay

10 self.day = gün

11 self.year = yıl

12 self.speakText = speakText

13

14 # Bu, içinde saklanan speakText'i döndüren bir erişimci yöntemidir.

15 # nesne. "self "in bir parametre olduğuna dikkat edin. Her metodun parametresi "self "tir.

16 # ilk parametre. "self" parametresi geçerli parametreye bir referanstır

17 # nesne. Geçerli nesne noktanın sol tarafında görünür (örn.

18 # yöntem çağrıldığında .).

19 **def** konuş(self):

20 **return** self.speakText

21

22 # İşte adı almak için bir accessor yöntemi

23 **def** getName(self):

24 **return** self.name

25

26 # Bu, doğum günü bilgisini kullanan başka bir erişimci yöntemidir

27 # tarihi temsil eden bir dize döndürür.

28  **def** birthDate(self):

29  **return** str(self.month) + "/" + str(self.day) + "/" + str(self.year)

30

31 # Bu, Dog nesnesinin speakText'ini değiştiren bir mutator yöntemidir.

32 **def** changeBark(self,bark):

33 self.speakText = bark

34

35 # Yeni yavru köpeği yaratırken doğum gününü bilmiyoruz. Seçin

36 # ilk köpeğin doğum günü artı bir yıl. Konuşma metni

37 # her iki köpeğin metninin birleştirilmesi. Sol taraftaki köpek +

38 # işleci "self" parametresi tarafından başvurulan nesnedir. Bu nesne

39 # "otherDog" parametresi + operatörünün sağ tarafındaki köpektir.

40 **def** \_\_add\_\_(self,otherDog):

41 **return** Dog("Puppy of " + self.name + " and " + otherDog.name, \

42 self.month, self.day, self.year + 1, \

43 self.speakText + otherDog.speakText)

44

45 **def** main():

46 boyDog = Dog("Mesa", 5, 15, 2004, "WOOOOF")

47 girlDog = Dog("Sequoia", 5, 6, 2004, "barkbark")

48 **print**(boyDog.speak())

49 **print**(girlDog.speak())

50  **print**(boyDog.birthDate())

51 **print**(girlDog.birthDate())

52 boyDog.changeBark("woofywoofy")

53 **print**(boyDog.speak())

54 puppy = boyDog + girlDog

55 **print**(puppy.speak())

56  **print**(puppy.getName())

57 **print**(puppy.birthDate())

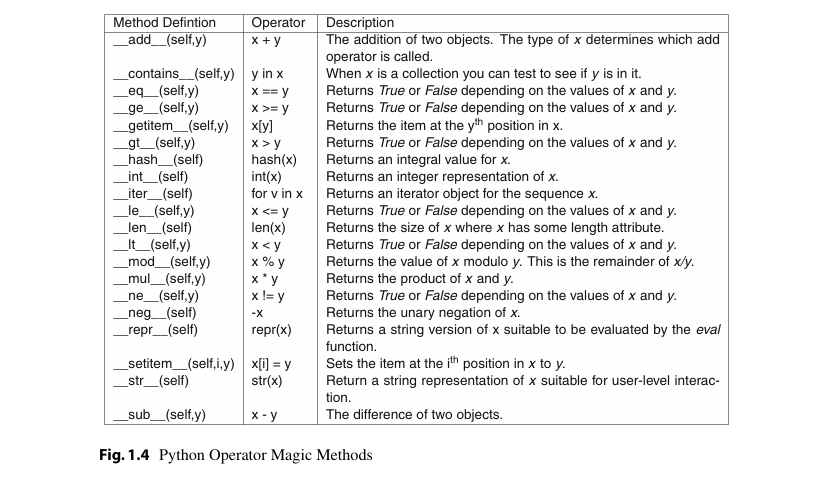
58

59 **if** \_\_name\_\_ == "\_\_main\_\_":

60 main()

Bu metin, operatörün aşırı yüklenmesini oldukça kapsamlı bir şekilde kullanmaktadır. Python programcıların bu operatörleri Sihirli Yöntemlerdir çünkü bir operatör bir operatörde kullanıldığında otomatik olarak çağrılır

Şekil 1.4'teki tabloda verilen yaygın operatörlerden bazıları



kolaylık sağlar. Her operatör için sihirli yöntem verilir, operatörün nasıl çağrılacağı ve bunun kısa bir açıklaması da verilmiştir. Tabloda self ve x değerleri aynıdır. Tabloda x'in türü, her durumda hangi operatör yönteminin çağrılacağını belirler. repr(x) ve str(x) operatörleri biraz daha fazla açıklamayı hak eder. Her iki operatör işleçleri x'in bir dize gösterimini döndürür. Aradaki fark str işlecinin repr işleci çağrıldığında insan etkileşimine uygun bir dize döndürür değerlendirilebilecek bir dize gösterimine ihtiyaç duyulduğunda. Örneğin, eğer biz bu iki operatörü Dog sınıfı üzerinde tanımlamak isteseydik, repr yöntemi "Dog('Mesa', 5,15,2004, 'WOOOF')" dizesini döndürürken str işleci sadece köpeğin adını döndürür. repr operatörü çağrıldığında, dizeyi bir ifadesi daha sonra Python'daki eval fonksiyonu tarafından değerlendirilebilirken str işleci basitçe bir nesne için bir dize döndürür.

**1.6 İçe Aktarma Modülleri +**

Python'da programlar modüllere ayrılabilir. Tipik olarak, bir program yazdığınızda Python'da program yazarken başka birinin yazdığı kodu kullanacaksınız. Bu kod genellikle bir modül içinde sağlanır. Bir modülü kullanmak için onu içe aktarırsınız. Bir modülü içe aktarmanın iki yolu vardır. Bu yazıda geliştirmekte olduğumuz çizim programı için bölümünde Turtle grafiklerini kullanmak istiyoruz. Turtle grafikleri ilk olarak uzun bir zaman önce Logo adlı bir programlama dili için tasarlandı. Logo 1967 civarında oluşturuldu Bu yüzden Turtle grafiklerinin temeli Bilgisayar Bilimleri açısından oldukça eskidir. Hala Bilgisayar Grafikleri hakkında düşünmenin yararlı bir yolu olmaya devam etmektedir. Fikrin çıkış noktası, bir Turtle bir kumsalda dolaşıyor ve dolaşırken kuyruğunu kumda sürükleyerek bir iz bırakıyor arkasında. Bir Turtle ile yapabileceğiniz her şey Bölüm 18'de tartışılmıştır. Python'da bir modülü içe aktarmanın iki yolu vardır: uygun yol ve güvenli yol. Kodu içe aktarmak için hangi yolu seçeceğiniz kişisel bir tercih olabilir, ancak kodu içe aktarmanın uygun yöntemini kullanmakla ilgili bazı çıkarımlar vardır. Turtle modülünü içe aktarmanın uygun yolu aşağıdakileri yazmak olacaktır.

**from turtle import**

t = Turtle()

Bu kullanışlıdır, çünkü Turtle sınıfını kullanmak istediğinizde sadece Turtle yazmanız yeterli olacaktır, ancak tamamen güvenli değildir çünkü Turtle tanımlayıcısını kodunuzda başka hiçbir şey için kullanmadığınızdan emin olmak için. İçinde Aslında, turtle modülünün tanımladığı ancak sizin farkında olmadığınız başka tanımlayıcılar da olabilir. Kodunuzda kullanmamanız gereken tanımlayıcılar olacaktır. Güvenli yol turtle modülünü içe aktarmak aşağıdaki gibi olacaktır.

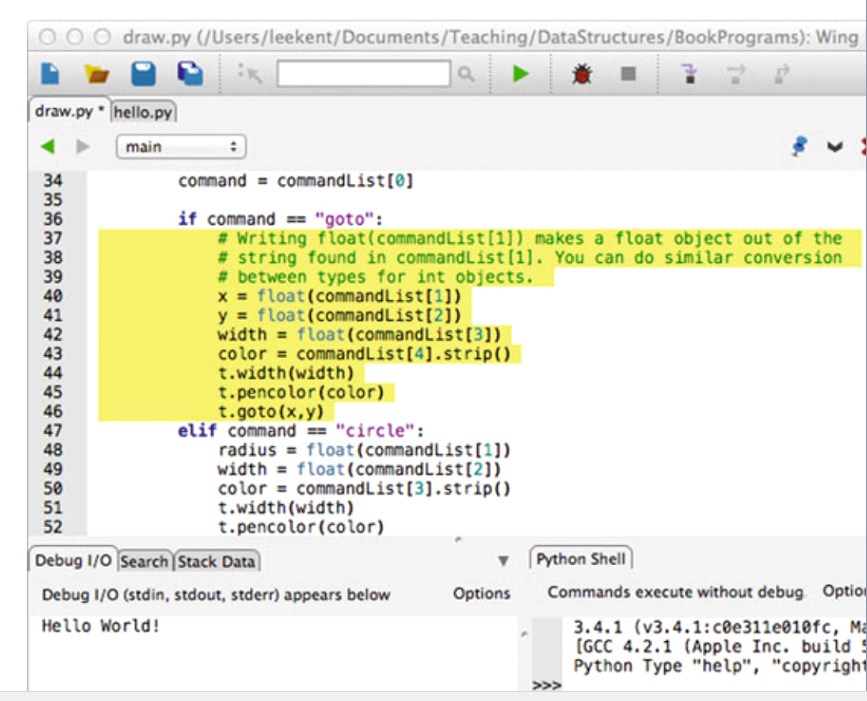
**import turtle**

t = turtle.Turtle()

Bu pek kullanışlı olmasa da, Turtle'dan önce "tur"modülünüzün ve Turtle modülünün ad alanı saklandığı için güvenlidir.Turtle modülündeki tüm tanımlayıcılar turtle isim alanında yer alırken yerel tanımlayıcılar yerel ad alanındadır. Bu isim alanları fikri önemli birçoğu programlama dilinin özelliğidir. Programcıların adım atmaktan kaçınmasına yardımcı olur. Bu metnin geri kalanında güvenli yöntem olan modülleri içe aktarma anlatılacaktır.

**1.7 Python Programlarında Girintileme**

Girinti, Python programlarında önemli bir rol oynar. Girintili bir satır, altında girintili olduğu satıra aittir. Bir fonksiyonun gövdesi, fonksiyon tanım satırının altında girintilidir. Bir if ifadesinin "then" kısmı, if ifadesinin altında girintilidir. Bir while döngüsünün gövdesi, döngü ifadesinin altında girintilidir. Bir sınıfın metodları, tümü sınıf tanım satırının altında girintilidir. Aynı miktarda girinti ile girintilenen ve bir araya toplanan tüm ifadeler “blok” olarak adlandırılır. Bir blok içindeki tüm ifadelerin tam olarak aynı miktarda girintili olması önemlidir. Eğer değil ise, Python tutarsız girinti hakkında şikâyette bulunur. Python için girinti o kadar önemli olduğundan, Wing IDE 101 size bir dizi satırı seçip bir grup olarak girintilerini ayarlamanıza olanak tanır, Şekil 1.5'te gösterildiği gibi. İlk olarak bloğun satırlarını seçersiniz ve ardından girintilerini artırmak için tab tuşuna basarsınız. Bir bloğun girintisini azaltmak için bloğun satırlarını seçersiniz ve ardından Shift-tab tuşuna basarsınız. Python kodu yazarken bu yaygın bir görevdir ve bir bloğun girintisini bir seferde ayarlayabilme yeteneği gerçekten zaman kazandırıcıdır.



**Şekil 1.5** Wing IDE 101'de Girintiyi Ayarlama

**1.8 Main Fonksiyon**

Programlar genellikle birçok fonksiyon tanımı ve fonksiyon çağrısı ile yazılır. Python'da bir işlev tanımı kurallara göre yazılır ve genellikle main fonksiyon olarak adlandırılır. Bu fonksiyon, programın ilk başlatıldığında tipik olarak çalıştırdığı kodları içerir. Bir Python programının genel taslağı Bölüm 1.8.1'de verilmiştir.

**1.8.1 Python Program Yapısı**

1 # importlar en başta yer alır.

2 **import turtle**

3

4 # diğer fonksiyon tanımları ana fonksiyon tanımından sonra gelir

5 **def** main ():

6 # Programın ana kodu buraya gelir.

7 t = turtle.Turtle()

8

9 # bu kod, her şeyin başlatılması için ana fonksiyonu çağırır. Bu ifadeye ilişkin koşul,

10 # if ifadesi, modül yürütücü tarafından çalıştırıldığında True değerini alır, ancak

11 # başka bir modüle aktarıldığında ise geçerli değildir.

12 **If** \_\_name\_\_ == "\_\_main\_\_":

13 main ()

Bölüm 1.8.1'deki kodun sonundaki if ifadesi, import ifadelerinden sonra çalıştırılan ilk koddur. Bu if ifadesinin koşulu, programın bağımsız bir program olarak çalıştırıldığında True değerini alır. Bazı durumlarda, başka bir modüle aktarmak isteyebileceğimiz modüller yazabiliriz. Bu if ifadesini ana fonksiyonu çağırmak için yazmak, modülün bağımsız bir program olarak çalıştırıldığında kendi ana fonksiyonunu çalıştırmasını sağlar. Modül başka bir modüle aktarıldığında ana fonksiyonunu çalıştırmaz. Daha sonra başka bir modüle aktarılmak üzere bir modül yazma fırsatınız olacak, bu nedenle bu şekilde her zaman ana fonksiyonu çağırmak iyi bir alışkanlıktır.

**1.9 Dosyadan Okuma**

Çizim programımıza başlamak için, bir resmin bir dosyada depolandığını ve bu dosyanın program başlatıldığında okunmasını istediğimizi varsayalım. Dosyanın her satırının bir çizim komutu ve ilişkili veriyi içerdiğini varsayacağız. Basit tutacak ve giriş dosyasında şu şekilde görünen çizim komutlarına bağlı kalacağız:

**⦁** goto, x, y, width, color (git, x, y, genişlik, renk)

**⦁** circle, radius, width, color (daire, yarıçap, genişlik, renk)

**⦁** beginfill, color (dolguya başla, renk)

**⦁** endfill (dolgu bitiş)

**⦁** penup (kalem kaldır)

**⦁** pendown (kalem bitiş)

Dosyanın her satırı, gerekli bilgileri içeren bir kayıt içerecektir. Bu komutların doğru sırasını içeren bir dosya sağlayarak bir resim çizebiliriz. Bölüm 1.9.1'deki dosya, bir pikap kamyonunu tanımlayan kayıtları içerir.

**1.9.1 Tek Satır Kayıtlı Bir Metin Dosyası**

1beginfill, black

2circle, 20, 1, black

3endfill

4penup

5goto, 120, 0, 1, black

6pendown

7beginfill, black

8circle, 20, 1, black

9endfill

10penup

11goto, 150, 40, 1, black

12pendown

13beginfill, yellow

14goto, -30, 40, 1, black

15goto, -30, 70, 1, black

16goto, 60, 70, 1, black

17goto, 60, 100, 1, black

18goto, 90, 100, 1, black

19goto, 115, 70, 1, black

20goto, 150, 70, 1, black

21goto, 150, 40, 1, black

22endfill

Bölüm 1.9.1'deki dosyadaki kayıtları işlemek için, bu dosyanın satırlarını okuyan ve her kayıt için uygun turtle grafik komutlarını yapan bir Python programı yazabiliriz. Her kayıt (yani çizim komutu), Bölüm 1.9.1'de açıklanan dosya biçiminde kendi başına bir satırdadır, bu nedenle dosyayı for döngüsü kullanarak okuyabiliriz. Bölüm 1.9.2'nin kodu, bu komutları okur ve dosyadaki her kaydı işleyerek içerdiği resmi çizer.

**1.9.2 Tek Satır Kayıtların Okunması ve İşlenmesi**

1 # Bu, turtle grafik modülünü içe aktarır.

2  **import turtle**

3

4 # Ana fonksiyon, programın ana kodunun yazıldığı yerdir.

5 def main():

6 # Bu satır, kullanıcıdan bir giriş satırı okur.

7 filename = input("Please enter drawing filename: ")

8

9 # Çizim yapmak için bir Turtle Grafik penceresi oluşturun.

10 t = turtle.Turtle()

11 # Ekran programın sonunda kullanılır.

12 screen = t.getscreen()

13

14 # Bir sonraki satır, dosyayı "r" veya okuma için açar. "w" onu yazma için açar.

15 # "y" dosyayı yazmak için açar ve "a" dosyayı eklemek için açar (yani sonuna eklemek için).

16 # Bu programda sadece dosyayı okumakla ilgileniyoruz.

17 file = open(filename, "r")

18

19 # Aşağıdaki for döngüsü, dosyanın satırlarını tek tek okur

20 # ve döngünün gövdesini dosyanın her satırı için bir kez çalıştırır.

21 **for** line **in** file:

22

23 # strip yöntemi, satırın sonundaki yeni satır karakterini

24 # ve satırın başında veya sonunda olabilecek boşlukları kaldırır.

25 text = line.strip()

26

27 # Aşağıdaki satır, metin değişkenini parçalarına ayırır.

28 # Örneğin eğer metin bunu içeriyorsa"goto, 10, 20, 1, black" sonrasında

29 # Komut listesi bu şekilde olacaktır ["goto", "10", "20", "1", "black"] sonra da

30 # Metni ayır.

31 commandList = text.split(",")

32

33 # çizim komutunu al

34 command = commandList[0]

35 if command == "goto":

36 #float(commandList[1]) ifadesi,

37 #commandList[1]'de bulunan dizeden bir float nesnesi oluşturur.

38 #Benzer şekilde, int nesneleri için de tür dönüşümü yapabilirsiniz.

39 x = float(commandList[1])

40 y = float(commandList[2])

41 width = float(commandList[3])

42 color = commandList[4].strip()

43 t.width(width)

44 t.pencolor(color)

45 t.goto(x,y)

46 elif command == "circle":

47 radius = float(commandList[1])

48 width = float(commandList[2])

49 color = commandList[3].strip()

50 t.width(width)

51 t.pencolor(color)

52 t.cirle(radius)

53 elif command == "beginfill":

54 color = commandList[1].strip()

55 t.fillcolor(color)

56 t.begin\_fill()

57 elif command == "endfill":

58 t.end\_fill(),

59 elif command == "penup":

60 t.penup()

61 elif command == "pendown":

62 t.pendown()

63 else:

64 print("Dosyada bilinmeyen komut bulundu:",command)

65 # dosyayı kapat

66 file.close()

67 # çizimi yapmak için kullandığımız turtle'ı gizleyin.

68 t.ht()

69 # Bu, programın turtle grafik penceresini fare tıklanana kadar açık tutmasına neden olur.

70 screen.exitonclick()

71 print("Program Execution Completed.")

72 # Bu kod, her şeyin başlatılması için ana fonksiyonu çağırır.

73 if \_\_name\_\_ == "\_\_main\_\_":

74 main()

Dosyanın her satırının kendi ayrı bir kaydı olduğu bir veri dosyanız olduğunda, bu kayıtları Bölüm 1.9.2'de yaptığımız gibi işleyebilirsiniz. Genel kalıp, dosyayı açmak, bir for döngüsü kullanarak dosyayı dolaşmak ve for döngüsünün gövdesinin her kaydı işlemesidir. Bölüm 1.9.3'teki sözde kod (pseudo-code), bir dosyadan tek satırlık kayıtları okumak için soyut bir kalıptır.

**1.9.3 Bir Dosyadan Tek Satırlık Kayıtları Okuma Modeli**

1 # ilk olarak dosya açılmalıdır.

2 file = open(filename,"r")

3

4 # for döngüsünün gövdesi dosyadaki her satır için bir kez çalıştırılır.

5 for line in file:

6 # Dosyanın her bir kaydını işleyin. Her kayıt, dosyanın tam olarak bir satırı olmalıdır.

7 # giriş dosyası. Bir kaydın işlenmesinin ne anlama geldiği,

8 # yazdığınız program tarafından belirlenecektir.

9 print(line)

10

11 # dosyayı kapatmak her zaman iyi bir fikirdir, ancak açıkça kapatmazsanız,

12 # programınız sonlandığında dosya da kapanacaktır.

13 file.close()

**1.10 Bir Dosyadan Çok Satırlı Kayıtları Okuma**

Bazen bir dosyanın kayıtları satır başına bir tane değildir. Bir dosyanın kayıtları birden fazla olabilir. Bu durumda, dosyayı okumak için bir for döngüsü kullanamazsınız. Bunun yerine bir while döngüsüne ihtiyacınız vardır. Eğer dosyayı okumayı bitirdiyseniz, while döngüsü kullandığınızda bir koşulu görebilmeniz için durumu kontrol edebilmeniz gerekir. Ancak, durumu kontrol etmek için önce bir kaydın en azından bir kısmını okumaya çalışmalısınız. Bu bir tür tavuk ve yumurta problemidir. Hangisi önce geldi, tavuk mu yumurta mı? Bilgisayar programcıları, dosyalardan okumayla ilgili bu sorun için bir isim bulmuşlardır. Buna Döngü ve Yarım Modeli (Loop and a Half Pattern) denir. Bir dosyayı okumak amacıyla while döngüsünü kullanmak için bir buçuk (bir tam ve bir yarım) döngüye ihtiyacımız vardır. Yarım, while döngüsünden önce gelir.

Bu bölümde yazdığımız programı düşünün. Kayıtların, dosyanın birden fazla satırında geçtiğini varsayalım. Hatta, kayıtların değişken uzunluğa sahip olduğunu varsayalım. Yani dosyamızın kayıtları bir ila beş satırdan oluşmaktadır. Çizim komutları aynen daha önce olduğu gibi olacaktır. Ancak, bir veri için tüm veri kayıtları tek bir satırda göründüğünde, Bölüm 1.10.1 'de gösterildiği gibi her bir veri parçasını kendi ayrı satırına koyacağız.

**1.10.1 Birden Fazla Satır Kaydı Olan Bir Metin Dosyası**

1 beginfill

2 black

3 circle

4 20

5 1

6 black

7 endfill

8 penup

9 goto

10 120

11 0

12 1

13 black

14 pendown

15 beginfill

16 black

17 circle

18 20

19 1

20 black

21 endfill

22 penup

23 goto

24 150

25 40

26 1

27 black

28 pendown

29 beginfill

30 yellow

31 goto

32 -30

33 40

34 1

35 black

36 goto

37 -30

38 70

39 1

40 black

41 goto

42 60

43 70

44 1

45 black

46 goto

47 60

48 100

49 1

50 black

51 goto

52 90

53 100

54 1

55 black

56 goto

57 115

58 70

59 1

60 black

61 goto

62 150

63 70

64 1

65 black

66 goto

67 150

68 40

69 1

70 black

71 endfill

Bölüm 1.10.1 'de gösterildiği gibi bir dosyayı okumak için, her kaydın ilk satırını okumak üzere döngümüzü yazar ve sonra bu satırı (yani grafik komutunu) kontrol ederiz böylece kaç satır daha okunacağını biliriz. Bölüm 1.10.2 'deki kod, bu değişken uzunluktaki kayıtları okumak için bir while döngüsü kullanır.

**1.10.2 Çok Satırlı Kayıtları Okuma ve İşleme**

1 import turtle

2

3 def main():

4 filename = input("Please enter drawing filename: ")

5

6 t = turtle.Turtle()

7 screen = t.getscreen()

8

9 file = open(filename, "r")

10

11 # Burada işleri başlatmak için yarım bir döngümüz var. Burada önce,

12 # grafik komutumuzu okumak, dosyanın boş olup olmadığını belirlememizi sağlar.

13 command = file.readline().strip()

14

15 # Komut boşsa, dosyada başka komut kalmamış demektir.

16 while command != "":

17

18 # Şimdi kaydın geri kalanını okumalı ve işlemden geçirmeliyiz. Çünkü,

19 # kayıtlar değişken uzunlukta olduğu için, hangi kaydı okuyacağımızı belirlemek için bir if-elif kullanacağız.

20 #Sonra kayıt türü ve kaydı okuyup işleyeceğiz.

21

22 if command == "goto":

23 x = float(file.readline())

24 y = float(file.readline())

25 width = float(file.readline())

26 color = file.readline().strip()

27 t.width(width)

28 t.pencolor(color)

29 t.goto(x,y)

30 elif command == "circle":

31 radius = float(file.readline())

32 width = float(file.readline())

33 color = file.readline().strip()

34 t.width(width)

35 t.pencolor(color)

36 t.circle(radius)

37 elif command == "beginfill":

38 color = file.readline().strip()

39 t.fillcolor(color)

40 t.begin\_fill()

41 elif command == "endfill":

42 t.end\_fill()

43 elif command == "penup":

44 t.penup()

45 elif command == "pendown":

46 t.pendown()

47 else:

48 print("Unknown command found in file:",command)

49

50 # Bu hala while döngüsünün içindedir. Okumalıyız (denemeliyiz).

51 # Dosyadan bir sonraki komutu okur. Okuma başarılı olursa, komut,

52 # boş dize olmayacak ve döngü tekrarlanacaktır. Eğer

53 # komut boşsa, bunun nedeni dosyada başka komut olmamasıdır

54 # dosya ve while döngüsü sonlandırılır.

55 command = file.readline().strip()

56

57

58 # dosyayı kapat

59 file.close()

60

61 t.ht()

62 screen.exitonclick()

63 print("Program Execution Completed.")

64

65 if \_\_name\_\_ == "\_\_main\_\_":

66 main()

Çok satırlı kayıtları olan bir dosyayı okurken while döngüsüne ihtiyaç vardır. Şuna dikkat edin 13. satırda while döngüsünden önce ilk kaydın ilk satırı okunur. Gövde için while döngüsünün yürütülmesi için, döngü yürütülmeden önce koşulun test edilmesi gerekir. Boş ya da değil dosyanın doğru olup olmadığını kontrol edebilmemiz için while döngüsünden önce bir satır okumak gerekir. Diğer her kaydın ilk satırı while döngüsünün sonunda okunur. 55. Satır döngü ve yarım modelidir. While döngüsünde ilk kaydın ilk satırı önce okunur, diğer tüm kayıtların ilk satırı while döngüsü içinde okunur. Koşul yanlış olduğunda while döngüsü sonlandırılır. Bir dosyadan çok satırlı kayıtları okumak için soyut model bölüm 1.10.3 ‘de gösterilir. Bu modelin kullanılabilecek başka biçimleri de vardır elbette ancak bu kalıbı ezberlemek zahmete değer çünkü bu kalıp hemen hemen her programlama dilinde işe yarayacaktır.

**1.10.3 Bir Dosyadan Çok Satırlı Kayıtları Okuma Modeli**

1 # İlk olarak dosya açılmalıdır.

2 file = open(filename, "r")

3

4 # Dosyadaki ilk kaydın ilk satırını okuyun. Elbette firstLine,

5 # programınızda anlamlı bir şekilde adlandırılmalıdır.

6 firstLine = file.readline().strip()

7

8 while firstLine != "":

9 # Kaydın devamını okuyun.

10 secondLine = file.readline().strip()

11 thirdLine = file.readline().strip()

12 # ...

13

14 # Daha sonra kaydı işleyin. Bu kullandığınız program tarafından belirlenecektir

15 # yazdığınız (program tarafından belirlenecektir.)

16 print(firstLine, secondLine, thirdLine)

17

18 # Son olarak, döngünün bir sonraki yinelemesini ayarlamak için

19 # bir sonraki kaydın ilk satırını okuyarak döngüyü bitirin.

20 firstLine = file.readline().strip()

21

22 # Dosyayı kapatmak iyi bir fikirdir, ancak otomatik olarak kapanacaktır,

23 # program sonlandırıldığında.

24 file.close()

**1.11 A CONTAINER CLASS**

Çizim programımızı daha da geliştirmek için öncelikle tüm çizim komutlarımızı tutacak bir veri yapısı oluşturacağız. Bu metinde kendi sınıfımızı tanımlamaya yönelik ilk örneğimiz olduğundan yavaş ilerleyeceğiz ve ne olduğu ve neden olduğu hakkında birçok ayrıntı sunacağız. Başlamak için bu konteyner sınıfıyla ne yapmak istediğimizi bulalım. Programımız boş bir konteyner oluşturarak başlayacak. Bunun için şu şekilde bir satır yazacağız.

graphicsCommands = PyList()

Daha sonra aşağıdaki gibi bir ekleme metodu kullanarak grafik komutlarını listemize eklemek isteyeceğiz.

command = GotoCommand(x, y, width, color)

graphicsCommands.append(command)

Ayrıca listemizdeki komutları yineleyebilmeyi de isteriz.

grafik Komutlarındaki komut için:

# t adlı turtle ı kullanarak ekrandaki her komutu çizin.

command.draw(t)

Bu noktada konteyner sınıfımız bir listeye çok benziyor. İlk veri yapısını göstermek ve bu ve sonraki bölümde listelerin nasıl verimli bir şekilde uygulanabileceğine dair tartışmayı motive etmek için kendi liste sınıfımızı tanımlıyoruz.

**1.12 POLIMORFIZM**

Nesneye Yönelik Programlamadaki önemli bir kavram polimorfizm olarak adlandırılır. Polimorfik kelimesi kelimenin tam anlamıyla birçok form anlamına gelir. Bu kavram bilgisayar programlamaya uygulandığında, fikir, belirli bir davranışın uygulanabileceği birçok yol olabileceğidir. PyList konteyner sınıfımızla ilişkili olarak,Oluştururken amaç, her tür grafik komutunun kendisini nasıl doğru şekilde çizeceğini bilmesidir. Örneğin, grafik komutlarından biri GoToCommand'dır. Bir GoToCommand çizildiğinde, ekranda geçerli noktadan bazı yeni (x,y) koordinatlara bir çizgi çizer. Ekranda ancak CircleCommand çizildiğinde belirli bir yarıçapa sahip daire çizer. Bu polimorfik davranış, her farklı davranış türü için bir sınıf ve çizim yöntemi oluşturularak tanımlanabilir. Bölümdeki kod. 1.12.1, farklı grafik çizim yöntemlerinin polimorfik davranışını tanımlayan sınıfların bir koleksiyonudur. Her çizim komutu için bir sınıf vardır.program tarafından işlenecektir.

**1.12.1 Grafik Komut Sınıfları**

1 # Aşağıdaki komut sınıflarının her biri, aşağıdakilerden biri için bilgi tutar

2 # bir grafik dosyasında bulunan komut türleri. Her bir komut için

3 # Bir **draw** metodu olmalı; bu metot bir **turtle** almalı ve **turtle** kullanarak çizim yapmalıdır.

4 # nesne. Her sınıf için bir draw metoduna sahip olarak

5 # bir dizi çizimi geçerken doğru çizim yöntemini polimorfik olarak çağırır

6 # bu komutlar. Polimorfizm, “doğru” çizim yöntemi

7 # hangi grafik komutunun çağrıldığını bilmek zorunda kalmadan çağrılır.

8  **class GoToCommand**:

9 # Burada yapıcı, genişlik ve renk için varsayılan değerlerle tanımlanmıştır.

10 # Bu, bir GoToCommand nesnesini GoToCommand(10,20) şeklinde oluşturabileceğimiz anlamına gelir,

11 # veya GoToCommand(10,20,5), veya GoToCommand(10,20,5, “yellow”).

12 **def** \_\_init\_\_(self,x,y,width=1,color="black"):

13 self.x = x

14 self.y = y

15 self.color = color

16 self.width = width

17

18 **def** draw(self,turtle):

19 turtle.width(self.width)

20 turtle.pencolor(self.color)

21 turtle.goto(self.x,self.y)

22

23 **class CircleCommand**:

24 **def** \_\_init\_\_(self,radius, width=1,color="black"):

25 self.radius = radius

26 self.width = width

27 self.color = color

28

29 **def** draw(self,turtle):

30 turtle.width(self.width)

31 turtle.pencolor(self.color)

32 turtle.circle(self.radius)

33

34 **class BeginFillCommand**:

35 **def** \_\_init\_\_(self,color):

36 self.color = color

37

38 **def** draw(self,turtle):

39 turtle.fillcolor(self.color)

40 turtle.begin\_fill()

41

42 **class EndFillCommand**:

43 **def** \_\_init\_\_(self):

44 # pass bir ifade yer tutucusudur ve hiçbir şey yapmaz. Hiçbir şeyimiz yok.

45 # bu sınıfta başlatmak için çünkü tek istediğimiz polimorfik

46 # draw yönteminin davranışı.

47 **pass**

48

49 **def** draw(self,turtle):

50 turtle.end\_fill()

51

52 **class PenUpCommand**:

53 **def** \_\_init\_\_(self):

54 **pass**

55

56 **def** draw(self,turtle):

57 turtle.penup()

58

59 **class PenDownCommand**:

60 **def** \_\_init\_\_(self):

61 **pass**

62

63 **def** draw(self,turtle):

64 turtle.pendown()

**1.13 ACCUMULATOR MODELI**

Az önce tanımladığımız farklı komut sınıflarını kullanmak için programımız, daha önce olduğu gibi **loop-and-a-half** (yarım döngü) desenini kullanarak dosyadan değişken uzunluktaki kayıtları okuyacaktır. **Programlama kalıpları**, bazen **idiom** olarak adlandırılır ve Bilgisayar Bilimi'nde önemli bir yere sahiptir. Bir **idiom** öğrendiğimizde, onu programlarımızda tekrar tekrar uygulayabiliriz. Bu bizim için faydalıdır çünkü bir problemi çözerken “Evet, bu problemi şu idiom'u kullanarak çözebilirim” diyebilmek işimizi kolaylaştırır. **Idioms** (programlama kalıpları), zihnimizi daha karmaşık problemlere odaklamamıza yardımcı olur.

Programlamada önemli kalıplardan biri **Accumulator Pattern** (Toplayıcı Deseni) olarak adlandırılır. Bu desen, yazdığımız hemen her programda kullanılır. Bu deseni kullanırken, bir döngüden önce bir **toplayıcı (accumulator)** başlatılır ve döngü içinde bu toplayıcıya değerler eklenir. Örneğin, **Bölüm 1.13.1**'de yer alan kod, **Accumulator Pattern**'i kullanarak 1'den 10'a kadar olan sayıların karelerini içeren bir liste oluşturur.

**1.13.1 SQUARES LİSTESİ**

1 # **Accumulator**'ı başlatın, bu durumda bir **list**.

2 accumulator = []

3

4 # Bir tür **for loop** veya **while loop** yazın.

5 **for** i **in** range(1,11):

6 # **Accumulator**'a ekleyin, bu durumda **list**'e ekleyin.

7 accumulator = accumulator + [i \*\* 2]

Grafik programımızı tamamlamak için, dosyadan kayıtları okumak üzere **loop-and-a-half** desenini ve her bulduğumuz kayıt için bir **command object**'i **PyList container**'ına eklemek amacıyla **accumulator pattern**'i kullanacağız. Kod **Bölüm 1.13.2**'de verilmiştir.

**1.13.2 A GraphicsProgram**

1 **import turtle**

2

3 # Komut sınıfları buraya eklenebilirdi ancak bunlar dışarıda bırakılmıştır çünkü

4 # bölümün önceki kısımlarında tanımlanmıştır.

5

6 # Bu bizim PyList sınıfımız. Grafiklerimizin bir listesini tutar

7 # komutlar.

8

9 **class PyList**:

10 **def** \_\_init\_\_(self):

11 self.items = []

12

13 **def** append(self,item):

14 self.items = self.items + [item]

15

16 # Bu dizi üzerinde yineleme yapmak istiyorsak, özel bir yöntem tanımlarız

17 # **iter**(self) çağrıldı. Bu olmadan “builtins.TypeError:

18 # 'PyList' nesnesi yinelenebilir değil” hatası alırız.

19 # for cmd in seq:

20 # burada seq bu dizilerden biridir. Aşağıdaki verim bir

21 # dizinin öğesi ve for öğesinin yürütülmesini askıya alacaktır.

22 # bir sonraki elemana ihtiyaç duyulana kadar aşağıdaki yöntemde döngü. Yetenek

23 # dizinin her bir elemanını gerektiği gibi vermek “tembel” değerlendirme olarak adlandırılır

24 # ve çok güçlüdür. Bu, yalnızca aşağıdaki gibi erişim sağlamamız gerektiği anlamına gelir

25 # dizinin gerekli olduğu kadar çok sayıda öğesi ve daha fazlası değil.

26 **def** \_\_iter\_\_(self):

27 **for** c **in** self.items:

28 **yield** c

29

30 **def** main():

31 filename = input("Please enter drawing filename: ")

32

33 t = turtle.Turtle()

34 screen = t.getscreen()

35 file = open(filename, "r")

36

37 # Grafik komutlarını tutmak için bir PyList oluşturun.

38 # dosyadan okuyun.

39 graphicsCommands = PyList()

40

41 command = file.readline().strip()

42

43 **while** command != "":

44

45 # Şimdi kaydın geri kalanını okumalı ve işlemden geçirmeliyiz. Çünkü

46 # kayıtlar değişken uzunlukta ise, hangi kayıtların değişken uzunlukta olduğunu belirlemek için bir if-elif kullanacağız.

47 # kayıt türüdür ve sonra kaydı okuyup işleyeceğiz.

48 # Bu programda, kaydın işlenmesi bir komut nesnesi oluşturulması anlamına gelir

49 # yukarıdaki sınıflardan birini kullanarak ve ardından bu nesneyi

50 # graphicsCommands PyList nesnesi.

51

52 **if** command == "goto":

53 x = float(file.readline())

54 y = float(file.readline())

55 width = float(file.readline())

56 color = file.readline().strip()

57 cmd = GoToCommand(x,y,width,color)

58

59 **elif** command == "circle":

60 radius = float(file.readline())

61 width = float(file.readline())

62 color = file.readline().strip()

63 cmd = CircleCommand(radius,width,color)

64

65 **elif** command == "beginfill":

66 color = file.readline().strip()

67 cmd = BeginFillCommand(color)

68

69 **elif** command == "endfill":

70 cmd = EndFillCommand()

71

72 **elif** command == "penup":

73 cmd = PenUpCommand()

74

75 **elif** command == "pendown":

76 cmd = PenDownCommand()

77 **else**:

78 # Bir istisnanın yükseltilmesi programı derhal sonlandıracaktır

79 # bilinmeyen bir şeyle karşılaştığımızda olmasını istediğimiz şey

80 # komutu. RuntimeError istisnası yaygın bir istisnadır

81 # yükseltmek için. İstisna olduğunda dize yazdırılacaktır

82 numara basıldı.

83 **raise** RuntimeError("Unknown Command: " + command)

84

85 # Finish processing the record by adding the command to the sequence.

86 graphicsCommands.append(cmd)

87

88 # Read one more line to set up for the next time through the loop.

89 command = file.readline().strip()

90

91 # This code iterates through the commands to do the drawing and

92 # demonstrates the use of the \_\_iter(self)\_\_ method in the

93 # PyList class above.

94 **for** cmd **in** graphicsCommands:

95 cmd.draw(t)

96

97 file.close()

98 t.ht()

99 screen.exitonclick()

100 **print**("Program Execution Completed.")

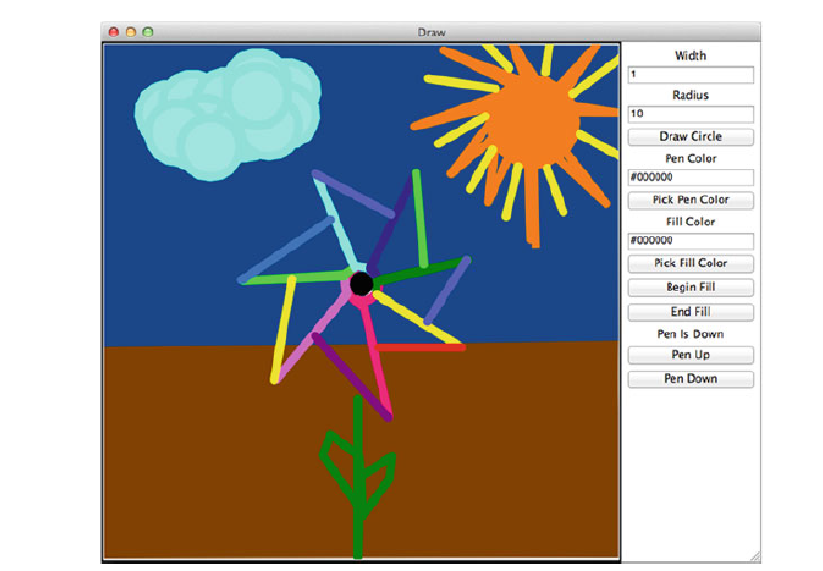
101

102 **if** \_\_name\_\_ == "\_\_main\_\_":

103 main()

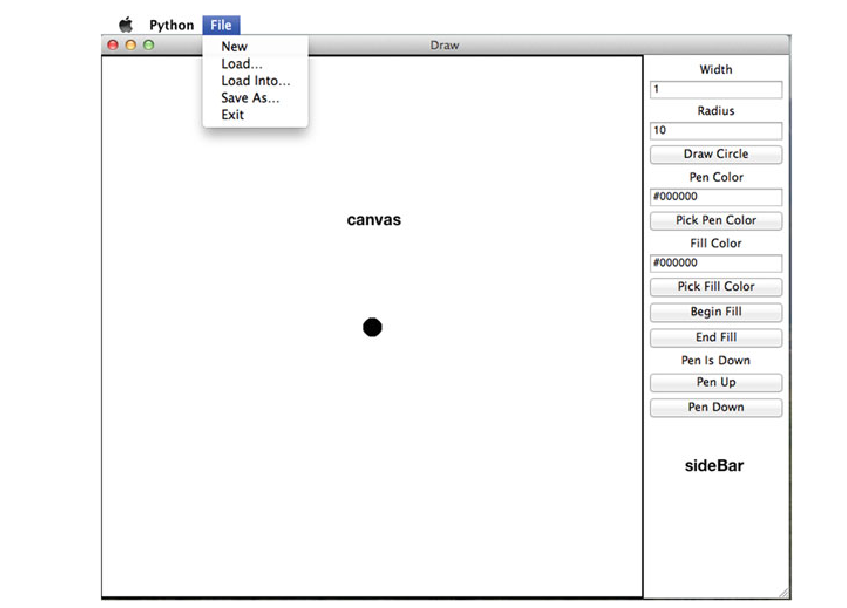
**d1.14 TKINTER ILE GUI UYGULAMAK**

G



B Bir **GUI** oluşturmak için bir pencere oluşturmanız gerekir. Bunu yapmak **Tkinter** kullanarak gerçekten çok basittir.

root = tkinter.Tk()  
Bu, ekranda boş bir pencere oluşturur, ancak tabii ki içine hiçbir şey koymaz. İçine **widgets** yerleştirmemiz gerekir, böylece **Şekil 1.6**'daki pencere gibi görünür (tabii ki Denise'in bizim için çizdiği güzel resmi hariç; teşekkürler Denise!). Ayrıca, çizim uygulamasındaki **events**'leri işlemek için **event handlers** oluşturmalıyız.**Widgets**'leri bir pencereye yerleştirmek **layout** olarak adlandırılır. Bir pencereyi düzenlemek için bir tür **layout manager** kullanılır. **Windowing toolkits** bazı türde bir düzenleme destekler. **Tkinter**'da **widgets**'leri bir pencereye ya **pack**, **grid** veya **place** yöntemiyle yerleştirirsiniz. **Widgets**'leri paketlerken, bu bir valiz paketlemeye benzer ve her **widget**, GUI'ye önceki **widget**'in yanına veya altına yerleştirilir. **Widgets**'leri paketlemek, çoğu durumda istediğiniz düzeni sağlar, ancak bazen bir pencereyi düzenlemek için **grid** faydalı olabilir. **Place layout manager**, **widgets**'leri bir pencere içinde belirli bir konuma yerleştirmenize olanak tanır. Çizim uygulamamızda düzenleme için **pack layout manager**'ı kullanacağız.**Widgets**'leri paketlerken, doğru düzeni elde etmek için bazen bir **Frame widget**'ı oluşturmanız gerekir. **Frame widgets**, diğer **widgets**'ları içinde tutar. **Şekil 1.7**'de iki **frame widget**'ı oluşturulmuştur. **DrawingApplication frame**, tüm pencerenin boyutundadır ve içinde sadece yan yana yerleştirilmiş iki **widget** vardır: **canvas** ve **sideBar frame**. **Canvas**, üzerinde bir **turtle**'ın çizebileceği bir **widget**'tır. **SideBar widget** ise tüm butonları, **entry boxes**'ları ve **labels**'ları tutar.



**DrawingApplication frame**, **Frame**'den türetilir. Nesne yönelimli bir dilde programlama yaparken bazen bir sınıfı uygulamak istersiniz, ancak bu sınıf neredeyse başka bir sınıfa benzer olur. Bu durumda, **DrawingApplication** bir **Frame**'dir. Bu, **DrawingApplication** nesnelerinin iki bölümden oluştuğu anlamına gelir: **DrawingApplication**'ın **Frame** kısmı ve geri kalan kısmı, bu durumda **PyList** dizisi olan **graphics commands**. Bizim **frame**'imiz, **canvas** üzerindeki resmi çizmek için kullanılan **graphics commands**'i takip edecektir. Kodun bazı kısımları **Bölüm 1.14.1**'de yer almaktadır. **Bölüm 1.14.1**'deki kod, oluşturulan tüm **widgets**'leri ve bunların pencereye nasıl paketlendiğini gösterir.**Canvas** ve **sideBar widgets**'leri **DrawingApplication frame**'ine yan yana eklenir. Ardından, tüm **entry**, **label** ve **button widgets**'leri **sideBar frame**'ine eklenir.Ayrıca, **Draw application**'da bir menü vardır. Menü, pencereye eklenen bir başka **widget**'tır (**self.master** olarak adlandırılır kodda, **Bölüm 1.14.1**'de). **fileMenu**, menü çubuğunda görünen kısmıdır. Menü öğeleri **“New”**, **“Load...”**, **“Load Into...”**, **“Save As...”** ve **“Exit”** bu menüye eklenir. Her menü öğesi, seçildiğinde çalıştırılan bir **event handler** ile bağlantılıdır.**Turtle object**'ı **Bölüm 1.14.1**'de oluşturulduğunda, bir **RawTurtle** olarak oluşturulur. **RawTurtle**, bir **turtle**'a çok benzer, ancak **RawTurtle**, üzerinde çizim yapması için bir **canvas** sağlanabilir. Bir **Turtle object**, ilk **turtle** oluşturulduğunda kendi **canvas**'ını oluşturur. Bizzaten **turtle** için bir **canvas**'a sahip olduğumuz için, bir **RawTurtle object** oluşturuyoruz. **Widgets** için **event handlers**'a ek olarak, üç başka **event handler** vardır. **onclick event**'i, fare düğmesine **canvas** üzerinde tıklanıldığında meydana gelir. **ondrag event handler**'ı, **turtle** **canvas** üzerindesürüklendiğinde çalışır. Son olarak, **undoHandler**, klavyede **u** tuşuna basıldığında çağrılır.

**1.14.1 A GUI Drawing Uygulaması**

1 # This class defines the drawing application. The following line says that

2 # the DrawingApplication class inherits from the Frame class. This means

3 # that a DrawingApplication is like a Frame object except for the code

4 # written here which redefines/extends the behavior of a Frame.

5 **class DrawingApplication**(tkinter.Frame):

6 **def** \_\_init\_\_(self, master=None):

7 super().\_\_init\_\_(master)

8 self.pack()

9 self.buildWindow()

10 self.graphicsCommands = PyList()

11

12 # This method is called to create all the widgets, place them in the GUI,

13 # and define the event handlers for the application.

14 **def** buildWindow(self):

15

16 # The master is the root window. The title is set as below.

17 self.master.title("Draw")

18

19 # Here is how to create a menu bar. The tearoff=0 means that menus

20 # can’t be separated from the window which is a feature of tkinter.

21 bar = tkinter.Menu(self.master)

22 fileMenu = tkinter.Menu(bar,tearoff=0)

23

24 # This code is called by the "New" menu item below when it is selected.

25 # The same applies for loadFile, addToFile, and saveFile below. The

26 # "Exit" menu item below calls quit on the "master" or root window.

27 **def** newWindow():

28 # This sets up the turtle to be ready for a new picture to be

29 # drawn. It also sets the sequence back to empty. It is necessary

30 # for the graphicsCommands sequence to be in the object (i.e.

31 # self.graphicsCommands) because otherwise the statement:

32 # graphicsCommands = PyList()

33 # would make this variable a local variable in the newWindow

34 # method. If it were local, it would not be set anymore once the

35 # newWindow method returned.

36 theTurtle.clear()

37 theTurtle.penup()

38 theTurtle.goto(0,0)

39 theTurtle.pendown()

40 screen.update()

41 screen.listen()

42 self.graphicsCommands = PyList()

43

44 fileMenu.add\_command(label="New",command=newWindow)

45

46 # The parse function adds the contents of an XML file to the sequence.

47 **def** parse(filename):

48 xmldoc = xml.dom.minidom.parse(filename)

49

50 graphicsCommandsElement = xmldoc.getElementsByTagName("GraphicsCommands")[0]

51

52 graphicsCommands = graphicsCommandsElement.getElementsByTagName("Command")

53

54 **for** commandElement **in** graphicsCommands: 55 **print**(type(commandElement))

56 command = commandElement.firstChild.data.strip()

57 attr = commandElement.attributes

58 **if** command == "GoTo":

59 x = float(attr["x"].value)

60 y = float(attr["y"].value)

61 width = float(attr["width"].value)

62 color = attr["color"].value.strip()

63 cmd = GoToCommand(x,y,width,color)

64

65 **elif** command == "Circle":

66 radius = float(attr["radius"].value)

67 width = float(attr["width"].value)

68 color = attr["color"].value.strip()

69 cmd = CircleCommand(radius,width,color)

70

71 **elif** command == "BeginFill":

72 color = attr["color"].value.strip()

73 cmd = BeginFillCommand(color)

74

75 **elif** command == "EndFill": 76 cmd = EndFillCommand()

77

78 **elif** command == "PenUp": 79 cmd = PenUpCommand()

80

81 **elif** command == "PenDown": 82 cmd = PenDownCommand()

83  **else**:

84  **raise** RuntimeError("Unknown Command: " + command)

85

86 self.graphicsCommands.append(cmd)

87

88 **def** loadFile():

89

90 filename = tkinter.filedialog.askopenfilename(title="Select a Graphics File")

91

92 newWindow()

93

94 # This re-initializes the sequence for the new picture.

95 self.graphicsCommands = PyList()

96

97 # calling parse will read the graphics commands from the file.

98 parse(filename)

99

100 **for** cmd **in** self.graphicsCommands: 101 cmd.draw(theTurtle)

102

103 # This line is necessary to update the window after the picture is drawn.

104 screen.update()

105

106

107 fileMenu.add\_command(label="Load...",command=loadFile)

108

109 **def** addToFile():

110 filename = tkinter.filedialog.askopenfilename(title="Select a Graphics File")

111

112 theTurtle.penup()

113 theTurtle.goto(0,0)

114 theTurtle.pendown()

115 theTurtle.pencolor("#000000")

116 theTurtle.fillcolor("#000000")

117 cmd = PenUpCommand()

118 self.graphicsCommands.append(cmd)

119 cmd = GoToCommand(0,0,1,"#000000")

120 self.graphicsCommands.append(cmd)

121 cmd = PenDownCommand()

122 self.graphicsCommands.append(cmd)

123 screen.update()

124 parse(filename)

125

126 **for** cmd **in** self.graphicsCommands: 127 cmd.draw(theTurtle)

128

129 screen.update() 130

131 fileMenu.add\_command(label="Load Into...",command=addToFile)

132

133 # The write function writes an XML file to the given filename

134 **def** write(filename):

135 file = open(filename, "w")

136 file.write(’<?xml version="1.0" encoding="UTF-8" standalone="no" ?>**\n**’) 137 file.write(’<GraphicsCommands>**\n**’)

138 **for** cmd **in** self.graphicsCommands:

139 file.write(’ ’+str(cmd)+"**\n**")

140

141 file.write(’</GraphicsCommands>**\n**’)

142

143 file.close()

144

145 **def** saveFile():

146 filename = tkinter.filedialog.asksaveasfilename(title="Save Picture As...")

147 write(filename)

148

149 fileMenu.add\_command(label="Save As...",command=saveFile)

150

151

152 fileMenu.add\_command(label="Exit",command=self.master.quit)

153

154 bar.add\_cascade(label="File",menu=fileMenu)

155

156 # This tells the root window to display the newly created menu bar.

157 self.master.config(menu=bar)

158

159 # Here several widgets are created. The canvas is the drawing area on

160 # the left side of the window.

161 canvas = tkinter.Canvas(self,width=600,height=600)

162 canvas.pack(side=tkinter.LEFT)

163

164 # By creating a RawTurtle, we can have the turtle draw on this canvas.

165 # Otherwise, a RawTurtle and a Turtle are exactly the same.

166 theTurtle = turtle.RawTurtle(canvas)

167

168 # This makes the shape of the turtle a circle.

169 theTurtle.shape("circle")

170 screen = theTurtle.getscreen()

171

172 # This causes the application to not update the screen unless

173 # screen.update() is called. This is necessary for the ondrag event

174 # handler below. Without it, the program bombs after dragging the

175 # turtle around for a while.

176 screen.tracer(0)

177

178 # This is the area on the right side of the window where all the

179 # buttons, labels, and entry boxes are located. The pad creates some empty

180 # space around the side. The side puts the sideBar on the right side of the

181 # this frame. The fill tells it to fill in all space available on the right

182 # side.

183 sideBar = tkinter.Frame(self,padx=5,pady=5)

184 sideBar.pack(side=tkinter.RIGHT, fill=tkinter.BOTH)

185

186 # This is a label widget. Packing it puts it at the top of the sidebar.

187 pointLabel = tkinter.Label(sideBar,text="Width")

188 pointLabel.pack()

189

190 # This entry widget allows the user to pick a width for their lines.

191 # With the widthSize variable below you can write widthSize.get() to get

192 # the contents of the entry widget and widthSize.set(val) to set the value

193 # of the entry widget to val. Initially the widthSize is set to 1. str(1) is

194 # needed because the entry widget must be given a string.

195 widthSize = tkinter.StringVar()

196 widthEntry = tkinter.Entry(sideBar,textvariable=widthSize)

197 widthEntry.pack()

198 widthSize.set(str(1))

199

200 radiusLabel = tkinter.Label(sideBar,text="Radius")

201 radiusLabel.pack()

202 radiusSize = tkinter.StringVar()

203 radiusEntry = tkinter.Entry(sideBar,textvariable=radiusSize)

204 radiusSize.set(str(10))

205 radiusEntry.pack()

206

207 # A button widget calls an event handler when it is pressed. The circleHandler

208 # function below is the event handler when the Draw Circle button is pressed.

209 **def** circleHandler():

210 # When drawing, a command is created and then the command is drawn by calling

211 # the draw method. Adding the command to the graphicsCommands sequence means the

212 # application will remember the picture.

213 cmd = CircleCommand(float(radiusSize.get()), float(widthSize.get()), penColor.get())

214 cmd.draw(theTurtle)

215 self.graphicsCommands.append(cmd)

216

217 # These two lines are needed to update the screen and to put the focus back

218 # in the drawing canvas. This is necessary because when pressing "u" to undo,

219 # the screen must have focus to receive the key press.

220 screen.update()

221 screen.listen()

222

223 # This creates the button widget in the sideBar. The fill=tkinter.BOTH causes the button

224 # to expand to fill the entire width of the sideBar.

225 circleButton = tkinter.Button(sideBar, text = "Draw Circle", command=circleHandler)

226 circleButton.pack(fill=tkinter.BOTH)

227

228 # The color mode 255 below allows colors to be specified in RGB form (i.e. Red/

229 # Green/Blue). The mode allows the Red value to be set by a two digit hexadecimal

230 # number ranging from 00-FF. The same applies for Blue and Green values. The

231 # color choosers below return a string representing the selected color and a slice

232 # is taken to extract the #RRGGBB hexadecimal string that the color choosers return.

233 screen.colormode(255)

234 penLabel = tkinter.Label(sideBar,text="Pen Color")

235 penLabel.pack()

236 penColor = tkinter.StringVar()

237 penEntry = tkinter.Entry(sideBar,textvariable=penColor)

238 penEntry.pack()

239 # This is the color black.

240 penColor.set("#000000")

241

242 **def** getPenColor():

243 color = tkinter.colorchooser.askcolor()

244 **if** color != None:

245 penColor.set(str(color)[-9:-2])

246

247 penColorButton = tkinter.Button(sideBar, text = "Pick Pen Color", command=getPenColor)

248 penColorButton.pack(fill=tkinter.BOTH)

249

250 fillLabel = tkinter.Label(sideBar,text="Fill Color")

251 fillLabel.pack()

252 fillColor = tkinter.StringVar()

253 fillEntry = tkinter.Entry(sideBar,textvariable=fillColor)

254 fillEntry.pack()

255 fillColor.set("#000000")

256

257 **def** getFillColor():

258 color = tkinter.colorchooser.askcolor()

259 **if** color != None:

260 fillColor.set(str(color)[-9:-2])

261

262 fillColorButton = \

263 tkinter.Button(sideBar, text = "Pick Fill Color", command=getFillColor)

264 fillColorButton.pack(fill=tkinter.BOTH)

265

266

267 **def** beginFillHandler():

268 cmd = BeginFillCommand(fillColor.get())

269 cmd.draw(theTurtle)

270 self.graphicsCommands.append(cmd)

271

272 beginFillButton = tkinter.Button(sideBar, text = "Begin Fill", command=beginFillHandler)

273 beginFillButton.pack(fill=tkinter.BOTH)

274

275 **def** endFillHandler():

276 cmd = EndFillCommand()

277 cmd.draw(theTurtle)

278 self.graphicsCommands.append(cmd)

279

280 endFillButton = tkinter.Button(sideBar, text = "End Fill", command=endFillHandler)

281 endFillButton.pack(fill=tkinter.BOTH)

282

283 penLabel = tkinter.Label(sideBar,text="Pen Is Down") 284 penLabel.pack()

285

286 **def** penUpHandler():

287 cmd = PenUpCommand()

288 cmd.draw(theTurtle)

289 penLabel.configure(text="Pen Is Up")

290 self.graphicsCommands.append(cmd) 291

292 penUpButton = tkinter.Button(sideBar, text = "Pen Up", command=penUpHandler)

293 penUpButton.pack(fill=tkinter.BOTH)

294

295 **def** penDownHandler():

296 cmd = PenDownCommand()

297 cmd.draw(theTurtle)

298 penLabel.configure(text="Pen Is Down")

299 self.graphicsCommands.append(cmd)

300

301 penDownButton = tkinter.Button(sideBar, text = "Pen Down", command=penDownHandler)

302 penDownButton.pack(fill=tkinter.BOTH)

303

304 # Here is another event handler. This one handles mouse clicks on the screen.

305 **def** clickHandler(x,y):

306 # When a mouse click occurs, get the widthSize entry value and set the width of the

307 # pen to the widthSize value. The float(widthSize.get()) is needed because

308 # the width is a float, but the entry widget stores it as a string.

309 cmd = GoToCommand(x,y,float(widthSize.get()),penColor.get())

310 cmd.draw(theTurtle)

311 self.graphicsCommands.append(cmd)

312 screen.update()

313 screen.listen()

314

315 # Here is how we tie the clickHandler to mouse clicks.

316 screen.onclick(clickHandler)

317

318 **def** dragHandler(x,y):

319 cmd = GoToCommand(x,y,float(widthSize.get()),penColor.get())

320 cmd.draw(theTurtle)

321 self.graphicsCommands.append(cmd)

322 screen.update()

323 screen.listen()

324

325 theTurtle.ondrag(dragHandler)

326

327 # the undoHandler undoes the last command by removing it from the

328 # sequence and then redrawing the entire picture.

329 **def** undoHandler():

330 **if** len(self.graphicsCommands) > 0:

331 self.graphicsCommands.removeLast()

332 theTurtle.clear()

333 theTurtle.penup() 334

theTurtle.goto(0,0)

335 theTurtle.pendown()

336 **for** cmd **in** self.graphicsCommands: 337 cmd.draw(theTurtle)

338 screen.update()

339 screen.listen() 340

341 screen.onkeypress(undoHandler, "u")

342 screen.listen()

343

344 # The main function in our GUI program is very simple. It creates the

345 # root window. Then it creates the DrawingApplication frame which creates

346 # all the widgets and has the logic for the event handlers. Calling mainloop

347 # on the frames makes it start listening for events. The mainloop function will

348 # return when the application is exited.

349 **def** main():

350 root = tkinter.Tk()

351 drawingApp = DrawingApplication(root)

352

353 drawingApp.mainloop()

354 **print**("Program Execution Completed.")

355

356 **if** \_\_name\_\_ == "\_\_main\_\_":

357 main()

**1.15 XML Dosyaları**

Döngüyü kullanarak okuduğumuz grafik komutları dosyası gibi standart bir metin dosyasını okuma Ve Bölüm 1.13.2'deki yarım desen, bilgisayar programlarında yaygın bir görevdir. Sadece sorun, programın girdinin belirli formatını okumak için yazılması gerektiğidir dosya. Eğer daha sonra girdi dosyasının formatını değiştirmek istersek, örneğin, Bir daire için dolgu rengi gibi yeni bir seçenek olursa , o zaman programı güncellemek zorunda kalırız ve bir kez okuduğu tüm dosyaları güncellemek gerekir. Giriş dosyası formatı ve program her zaman senkronize edilmelidir. Bu, tüm eski biçimlendirilmiş girdi dosyalarının yeni formata uygun hale getirilmeli ya da atılmalıdır. Bu çoğu kişi için kabul edilebilir bir durum değildir. çünkü veriler değerlidir. Bu sorunla başa çıkmak için bilgisayar programcıları bir dil tasarladılar. XML olarak adlandırılan ve açılımı eXtensible Markup Language olan veri giriş dosyalarını tanımlar. XML, veri açıklaması için bir meta dildir. Diğer dilleri tanımlamak için meta bir dildir. XML meta-dili evrensel olarak kabul görmüştür. Aslında, XML formatı bir standartlar komitesi tarafından yönetilir, bu da şu anlama gelir XML formatının çok kararlı ve geriye dönük olarak uyumlu kalacağına sonsuza dek güvenebiliriz. XML'e yapılacak herhangi bir eklemenin halihazırda mevcut olanlarla uyumlu olması gerekecektir. Bir XML belgesi, onu bir XML dosyası olarak tanımlamak için özel bir satırla başlar. Bu satır şöyle görünür.



Bir XML dosyasının geri kalanı öğelerden veya düğümlerden oluşur. Her düğüm şu şekilde tanımlanır bir etiket veya bir çift başlangıç ve bitiş etiketi. Her etiket sınırlandırılır (yani çevrelenir) açılı parantezler tarafından. Örneğin, işte böyle bir etiket.



Etiket adının hemen önündeki eğik çizgi, bunun bir kapanış etiketi olduğu anlamına gelir. Bir açılış ve kapanış etiketi iki etiket arasında metin veya başka XML öğeleri içerebilir, bu nedenle XML belgeleri, aşağıdakilere bağlı olarak gerektiği kadar derin iç içe geçmiş XML öğeleri içerebilir kodlamaya çalıştığınız veri. Her XML öğesinin kendisiyle ilişkili öznitelikleri olabilir. Örneğin, şunları düşünün bir GoTo grafiği yapmak için gereken bilgileri kapsayan bir XML öğesi komutuna ihtiyacımız var. Bir GoTo komutunu tamamlamak için x ve y koordinatları, genişliği ve kalem rengi. İşte bu bilgilerin kodlanmasına bir örnek XML formatı.



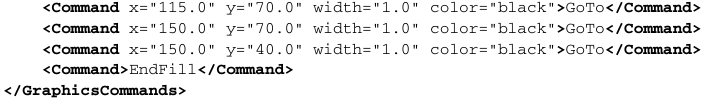
Etiket adının hemen önündeki eğik çizgi, bunun bir kapanış etiketi olduğu anlamına gelir. Bir açılış ve kapanış etiketi iki etiket arasında metin veya başka XML öğeleri içerebilir, bu nedenle XML belgeleri, aşağıdakilere bağlı olarak gerektiği kadar derin iç içe geçmiş XML öğeleri içerebilir kodlamaya çalıştığınız veri. Her XML öğesinin kendisiyle ilişkili öznitelikleri olabilir. Örneğin, şunları düşünün bir GoTo grafiği yapmak için gereken bilgileri kapsayan bir XML öğesi komutuna ihtiyacımız var. Bir GoTo komutunu tamamlamak için x ve y koordinatları, genişliği ve kalem rengi. İşte bu bilgilerin kodlanmasına bir örnek XML formatı.



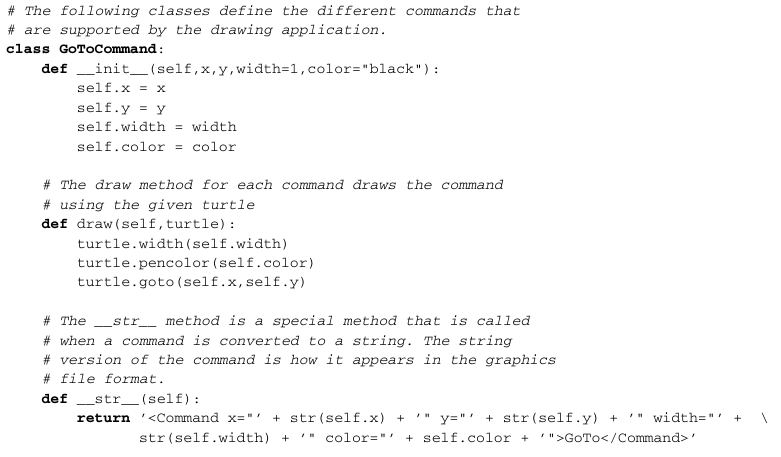
Bu örnekte, öznitelikler x, y, genişlik ve renktir. Her öznitelik, yukarıda gösterildiği gibi değeriyle eşlenir. GoTo metni, açılış ve kapanış etiketleri arasında görünen metindir. Bu metin bazen alt veri olarak adlandırılır. Tüm grafik komutları girdi dosyasını XML biçiminde kodlayarak, aşağıdaki sorunları ortadan kaldırırız Draw programı ve verileri arasındaki bağımlılığın bir kısmı. XML formatı (yani gramer) dışında, XML dosyasının içeriği tamamen programcıya veya veriyi kullanan programcılara bağlıdır. Bölüm 1.15.1'deki dosya kamyon resminin XML girdi dosyasının bir örneğidir.

**1.15.1 TheTruckXMLFile**





XML dosyaları metin dosyalarıdır. Sadece XML dosyalarının nasıl okunacağını standartlaştırmak için ek XML formatlı veriler içerirler. Bir XML dosyası yazmak, bir metin dosyası yazmak kadar basittir. XML dosyalarında girinti zorunlu olmamakla birlikte, genellikle vurgulamak için kullanılır. Dosyanın formatını belirler. Bölüm 1.15.1'de, GraphicsCommands elemanı bir adet Görüntüdeki her çizim komutu için komut öğesi. Her çizim komutu, metin olarak komut türünü içerir. Komut türleri GoTo, Circle, BeginFill, EndFill, PenUp ve PenDown'dan oluşur. Bir komutun nitelikleri, çeşitli komut türleri tarafından kullanılan x, y, genişlik, yarıçap ve renk gibi verilerdir. Komutları bir dosyaya yazmak için, Command sınıflarının her biri, özel \_\_str\_\_ yöntemi kullanılarak bir dizeye dönüştürüldüğünde bir XML öğesi üretecek şekilde değiştirilebilir. Örneğin, Bölüm 1.15.2, bir XML öğesinin oluşturulmasını destekleyen değiştirilmiş GoToCommand sınıfını içerir.



**1.15.2 XML Oluşturma Kodlu GoToCommand**

1 # Aşağıdaki sınıflar farklı komutları tanımlar

2 # çizim uygulaması tarafından desteklenir.

3 class GoToCommand:

4 def \_\_init\_\_(self,x,y,width=1,color="black"):

5 self.x = x

6 self.y = y

7 self.width = genişlik

8 self.color = color

9

10 # Her komut için çizim yöntemi komutu çizer

11 # verilen kaplumbağayı kullanarak

12 def draw(self,turtle):

13 kaplumbağa.genişlik(self.genişlik)

14 turtle.pencolor(self.color)

15 turtle.goto(self.x,self.y)

16

17 # \_\_str\_\_ yöntemi, şu şekilde adlandırılan özel bir yöntemdir

18 bir komut dizeye dönüştürüldüğünde. Dize

19 # komutunun sürümü grafiklerde göründüğü şekildedir

20 # dosya formatı.

21 def \_\_str\_\_(self): return '<Komut x="' + str(self.x) + '" y="' + str(self.y) + '" width="' + \

23 str(self.width) + '" color="' + self.color + '">GoTo</Command>'

Komut nesnelerinin her birinden bunun gibi bir dize döndürerek, yazılacak kod

çizim programının verilerini bir dosyaya aktarmak çok basittir. Gerekli olan tek şey bazı kodlardır

XML satırını ilk satır olarak yazar, ardından <GraphicsCommands> etiketi ve

komut öğeleri. Son olarak, <GraphicsCommands> etiketi yazılmalıdır.

Bölüm 1.15.3'teki kod bunu gerçekleştirir.

**1.15.3 Grafik Komutlarını XML Dosyasına Yazma**

1 dosya = open(dosya adı, "w")

2 file.write('<?xml version="1.0" encoding="UTF-8" standalone="no" ?>\n')

3 file.write('<GraphicsCommands>\n')

4 for cmd in self.graphicsCommands:

5 file.write(' '+str(cmd)+"\n")

6

7 file.write('</GraphicsCommands>\n')

8 file.close()

XML dosyası yazmak herhangi bir metin dosyası yazmak gibidir, ancak metin dosyasının XML dil bilgisi belirtimine uygun olmalıdır. XML oluşturmanın kesinlikle yolları vardır. Bölüm 1.15.3'te sunulduğundan farklı olan dosyalar. Bir sonraki bölümde XML ayrıştırıcıları ve XML belgelerini okumanın çok basit bir yolunu öğrenin. Bu dönen bir XML belgesi yazmayı kolaylaştıran en azından bazı XML çerçeveleri vardır. aynı derecede basittir.

**1.16 XML Dosyalarını Okuma**

XML dosyalarını normal bir metni okuduğumuz gibi okumak zorunda olsaydık okumak zor olurdu dosya: Bu özellikle doğrudur çünkü XML dosyaları satır odaklı değildir. Uyum sağlarlar'daki satırlar hakkında hiçbir şey belirtmez, ancak gramer XML gramerine Dosya. Bir XML dosyasını dosyanın satırlarını okuyarak okumak yerine, özel bir ayrıştırıcı adı verilen bir araçtır. Bir ayrıştırıcı, bir gramerin kurallarına göre yazılır. bu durumda XML grameri: Yazılmış birçok XML ayrıştırıcı vardır ve farklı ayrıştırıcılar farklı özelliklere sahiptir. Bu metinde kullanacağımız bir minidom adı verilen daha basit ayrıştırıcılardan biridir. Minidom ayrıştırıcısı bir XML'in tamamını okur üzerinde parse yöntemini çağırarak dosya. Bir XML dosyasının tüm içeriğini yerleştirir bir dizi Element nesnesine dönüştürür. Bir Element nesnesi alt verileri içerir ve tanımlanabilecek diğer öğelerle birlikte bir XML öğesinin öznitelikleri bu öğenin içinde. Minidom ayrıştırıcısını kullanmak için önce minidom'un bulunduğu modülü içe aktarmalısınız. Parser tanımlanmıştır.

import xml.dom. Minidom Ardından, bir XML üzerinde parse yöntemini çağırarak tüm XML dosyasını okuyabilirsiniz aşağıdaki gibi belge.

xmldoc = xml.dom.minidom.parse(dosya adı)

Bunu yaptıktan sonra, XML'den belirli bir öğe türünü okuyabilirsiniz üzerinde getElementsByTagName yöntemini çağırarak dosya. Örneğin, almak için Grafik komutları XML dosyasındaki Graphics Commands öğesini kullanarak bunu yaz XML belgesi GraphicsCommands öğesini içerir. GraphicsCommands öğesinde getElementsByTagName çağrısı, eşleşen tüm öğelerin bir listesini döndürür bu etiket adı. Dosyada bu etiketlerden yalnızca bir tane olduğunu bildiğimiz için listeden ilk elemanı almak için [0] yazın. Ardından, graphicsCommands öğesi dosyadan sadece bir öğe ve dosyanın tüm Komut öğelerini içerir içinde yer alır. Tüm bu öğelerin üzerinden geçmek istiyorsak, bir for Bölüm 1.16.1'deki kodda olduğu gibi döngü.

**1.16.1 XML Ayrıştırıcı Kullanma**

1 for commandElement in graphicsCommands:

2 print(type(commandElement))

3 command = commandElement.firstChild.data.strip()

4 attr = commandElement.attributes

5 if command == "GoTo":

6 x = float(attr["x"].value)

7 y = float(attr["y"].value)

8 width = float(attr["width"].value)

9 color = attr["color"].value.strip()

10 cmd = GoToCommand(x,y,width,color)

11

12 elif command == "Circle":

13 radius = float(attr["radius"].value)

14 width = float(attr["width"].value)

15 color = attr["color"].value.strip()

16 cmd = CircleCommand(yarıçap,genişlik,renk)

17

18 elif command == "BeginFill":

19 color = attr["color"].value.strip()

20 cmd = BeginFillCommand(color)

21

22 elif command == "EndFill":

23 cmd = EndFillCommand()

24

25 elif command == "PenUp":

26 cmd = PenUpCommand()

27

28 elif command == "PenDown":

29 cmd = PenDownCommand()

30 else:

31 raise RuntimeError("Bilinmeyen Komut: " + command)

32

33 self.append(cmd)

Bölüm 1.16.1'deki kodda attr değişkeni, özniteliği eşleyen bir sözlüktür adlarını (yani anahtarlarını) ilişkili değerlerine dönüştürür. Bir Komut düğümünün alt verileri şunlar olabilir düğüm için firstChild.data öğesine bakılarak bulunur. Strip yöntemi, düğümleri soymak için kullanılır dizede görünebilecek istenmeyen boşlukları, sekmeleri veya satır sonu karakterlerini ortadan kaldırır.

**1.17 Bölüm Özeti**

Bu ilk bölümde, aşağıdaki hususların dikkate alınması gereken geniş bir malzemeyi ele aldık Çoğunlukla inceleme ama muhtemelen sizin için de yeni olan bazı şeyleri kapsıyor. Yapma her şeyden çok bunalmış olabilirsiniz. Bu bölümün amacı size sorular sordurmaktır Anlamadığınız şeyler hakkında. Eğer bir şeyi anlamıyorsanız Öğretmeninize veya Python'da programlama hakkında daha fazla bilgisi olan birine sorun. Onlar muhtemelen size yardımcı olabilir. Soru sormak öğrenmek için harika bir yoldur ve Bilgisayar Bilimleri yaşam boyu öğrenme ile ilgilidir. İşte bu bölümde öğrenmiş olmanız gereken önemli kavramların bir listesi. Şunları yapmalısınız:

- Hem gerçek bir değerden hem de nesnenin açıkça kurucu.

- Python'da bir değeri (yani bir nesneyi) işaret eden referans kavramını anlamak.

- Bir nesne üzerinde bir yöntemin nasıl çağrılacağını bilir.

- Bir modülün nasıl içe aktarılacağını bilir.

- Python programlarında girintilemenin önemini anlamak.

- Python programlarında neden bir ana fonksiyon yazıldığını ve ana fonksiyonun nasıl çağrılacağını bilir.

- İster çok satırlı, ister tek satırlı, ister sabit olsun, bir dosyadan kayıtların nasıl okunacağını bilir

uzunlukta veya değişken uzunlukta kayıtlar.

- Bu bölümde tanımlanan PyList gibi bir konteyner sınıfının nasıl tanımlanacağını bilmek.

- Çok biçimlilik kavramını ve bunun bir nesnenin aşağıdakileri yapacağı anlamına geldiğini anlamak

Bir yöntem çağrıldığında doğru şeyi yapmak.

- Akümülatör modelini ve bir programda nasıl kullanılacağını anlamak.

- Python'da Tkinter kullanarak basit bir GUI'nin nasıl uygulanacağını bilir. Giriş kutuları, etiketler,

düğmeler, çerçeveler ve olay işleyicileri anlaşılan kavramlar olmalı ve

bu bölümdeki örneklere bakarak programlanabilir.

- Ve son olarak programlarınızda XML dosyalarını nasıl okuyup yazacağınızı bilmelisiniz.

Bu bölümde çok sayıda örnek kod bulunmaktadır ve Çizim'in son sürümü programı metnin web sitesinde veya Bölüm 20.1'de verilmiştir. Şüpheli olsa da bu bölümde bulduğunuz kodun her satırını ezberleyebilecek durumda olacaksanız, şunları yapmalısınız baktığınızda işlerin nasıl yürüdüğünü bildiğinizden emin olun ve hatırlamanız gerekir Bu bölümü bir kaynak olarak kullanabileceğinizi unutmayın. İhtiyaç duyduğunuzda sık sık geri dönün sonraki bölümlerde nasıl yapılacağını görebilirsiniz. Bu örnek kodu referans olarak kullanma gelecek bölümlerde sorularınızın çoğunu yanıtlamaya yardımcı olacaktır

**1.18 İnceleme Soruları**

1. IDE ne anlama gelir ve bir IDE kullanmak neden iyi bir fikirdir?

2. Mutlu Doğum kelimelerini içeren bir dizi oluşturmak için hangi kodu yazardınız? gün!? Yeni oluşturulan nesneye metin adı verilen bir referans kodu yazın.

3. Son soruda oluşturduğunuz dizeyi almak için hangi kodu yazardınız?

ve iki dizgi içeren bir listeye bölelim mi? lst referansını bu yeni listeye yönlendirin

oluşturulan liste.

4. İkinci sorgularda oluşturulan dizeyi büyük harfle yazmak için hangi kodu yazardınız?

tion. bDayWish adlı referansı bu büyük harfli dizeye yönlendirin.

5. Son üç soruyu yanıtlamak için yazdığınız kodu çalıştıracak olsaydınız,

Bu üç satır çalıştırıldıktan sonra text tarafından referans verilen dizin ne içerecektir kodun?

6. "Kent" kelimesini "Denise" kelimesiyle eşleştiren bir sözlüğü nasıl oluşturursunuz?

"Steve" mi "Lindy" mi? Bu iki durumda "Kent" ve "Steve" anahtarlardır ve

"Denise" ve "Lindy" değerlerdir.

7. Bölüm.17'ye başvurun. Önceki probda olduğu gibi bir anahtarı bir değere nasıl eşlersiniz?

sözlük ilk kez boş bir sözlük olarak oluşturulduğunda ne oldu? İPUCU: Bu

ekteki belgelerde bir öğenin ayarlanması olarak adlandırılacaktır. Yazmak

boş bir sözlük oluşturmak için kısa bir kod parçası ve ardından "Kent" ile

"Denise" ve "Steve", "Lindy "ye.

8. x < y yazıldığında hangi yöntem çağrılır? a metodu hangi sınıfta yer alır?

üye? Başka bir deyişle, bir programda size x < y gösterilirse, nasıl

tam olarak anlamak için hangi sınıfı incelemeniz gerektiğini bulabilir misiniz?

x < y ne anlama geliyor?

9. x y yazıldığında hangi yöntem çağrılır?

10. Döngü ve yarım problemi nedir ve nasıl çözülür?

11. Bir XML okumak için döngü ve yarım probleminin çözümünü kullanmanız gerekir mi? file? Neden ya da neden değil?

12. Polimorfizm ve Operatör Aşırı Yüklemesi birbiriyle yakından ilişkili kavramlardır. Yapabilir misiniz Bu iki kavramın nasıl benzer olduğunu ve Python'un bunları nasıl desteklediğini kısaca açıklayabilir misiniz? İPUCU: Hayır geçerli bir cevap değildir.

13. Bir programın kullanıcıdan bir tamsayı girmesini istemesi için ne yazardınız ve daha sonra 2'den kullanıcının belirttiği tamsayıya kadar olan tüm tamsayıları toplar. İPUCU: Bunu gerçekleştirmek için aralık fonksiyonunu nasıl kullanacağınızı gözden geçirmek isteyebilirsiniz. Hangi programlama modelini kullanabileceğinize karar verin.

14. Tkinter kullanarak nasıl bir pencere oluşturursunuz?

15. Bir Tkinter programında Frame nesnesinin amacı nedir?

16. Tkinter çerçevesindeki üç widget türü nedir?

17. Bir XML dosyasını okurken, dosyayı okumak için kaç satır kod gerekir?

18. Bir XML belgesinden öğe nasıl silinir? Hangi kod satırı(satırları) yazmak zorunda mısınız? Bir örnek verin

19. Bir XML belgesinde gezinirken, bu belgeden öğelerin bir listesini nasıl alırsınız? Hangi kod satır(lar)ını yazmak zorundasınız? Bir örnek veriniz.

20. Bir XML belgesindeki öznitelik nedir ve bir özniteliğe nasıl erişirsiniz? değer? Metinden veya bulduğunuz başka bir örnekten bir örnek veriniz.

**1.19 Programlama Problemleri**

1. Değişkenli bir giriş dosyasını okuyan Drawprogramının versiyonu ile başlayarak uzunluk kayıtları, programa seçtiğiniz yeni bir grafik komutunu ekleyin. Bir dosyaya nasıl yazılacağını düşünün, bir test dosyası oluşturun, kodunuzu yazın, ve test edin. İki dosya tasarlamalısınız: örnek bir test dosyası ve programın kendisi. Bazı örnekler, bir sayı ile bir yıldız çizmek için bir grafik komutu olabilir noktalar, yüksekliği ve genişliği olan bir dikdörtgen vb.

2. Bölüm 20.1'de verilen Çizim programından başlayarak, programı şu şekilde genişletin Draw programı için yeni bir şekil çizmek üzere yeni bir düğme ekleyin. Örneğin, çizim programının ekrana bir yıldız ya da gülen yüz gibi bir şey çizmesini sağlayın seçtiğiniz. İPUCU: Çiziminizi yapmak için ileri ve geri yöntemlerini kullanırsanız şeklini, her ileri ve geri miktarını bir ölçek ile çarparak ölçeklendirebilirsiniz değerini seçin. Ardından, kullanıcının bunun için bir ölçek seçmesine izin verebilirsiniz (veya yarıçap miktarını kullanabilirsiniz ölçeğiniz olarak) ve şeklinizi istediğiniz boyutta çizin. Bunu tamamlamak için alıştırması için XML formatınızı yeni bir grafik komutu içerecek şekilde genişletmeniz gerekir yeni şeklinizi çizmek için ilgili bilgileri saklamak üzere. Ayrıca şunları da yapmalısınız Yeni şekliniz için yeni bir graphicsCommand sınıfı tanımlayın.

3. Draw resmi üzerine bir metin dizesi çizme özelliğini ekleyin. Şuna izin vermeniz gerekecek kullanıcı bir punto boyutu seçsin. Gerçek bir meydan okuma için, kullanıcının yazı tipi türünü şuradan seçmesine izin verin yazı tipi türlerinden oluşan bir açılır liste. Kullanıcının girmesini istediğiniz bir dize çizin giriş kutusu.

4. İnternette seçtiğiniz bir XML belgesini bulun, kodunu yazın ve ve bu verilerden ister zaman içinde bir değer olsun, ister başka bir şey. Bulduğunuz verileri çizmek için kaplumbağa grafiklerini kullanın.

5. Bölüm 20.1'de sunulan çizim programına yeni bir düğme ekleyerek bir gökkuşağı kaplumbağanın mevcut konumunun üzerinde ortalanır. Bu oldukça yapılabilir sin ve cos (yani sinüs ve kosinüs) kullanılarak kolayca yapılabilir. Sin ve cos fonksiyonları şunları alır bir parametre olarak radyan. Bir gökkuşağı çizmek için radyan 0 ile math modülünden math.pi. Erişim için math modülünü içe aktarmanız gerekir math.cos ve math.sin'in yanı sıra math.pi için de kullanabilirsiniz. Değerleri bir yay içinde çizmek için aforloopandlet variable,i,rangefrom0to100.Thenradius\*math.cos(i/100.0 \* math.pi), yarıçap \* math.sin(i/100.0 \* math.pi), döngünün bir sonraki x,y koordinatıdır. gökkuşağının yayı. Yarıçapı değiştirerek gökkuşağınız için çeşitli çizgiler elde edeceksiniz. Her şerit farklı bir renge sahip olmalıdır. Rengi değiştirmek için, bir 24 bitlik sayıyı onaltılık sayıya dönüştürme. Python'da bir sayıyı hexadecimal'e dönüştürmek için şunları kullanabilirsiniz hex fonksiyonu. Renk dizenizin 6 basamak uzunluğunda olduğundan emin olmalısınız ve Python'da geçerli bir renk dizesi olması için bir pound işareti (yani #) ile başlar.

**1.** IDE ne anlama gelir ve bir IDE kullanmak neden iyi bir fikirdir?

Bazı avantajlar:

1. Verimlilik Artışı: IDE'ler, kod yazma sürecini hızlandıran bir dizi araç ve özellik sunar. Otomatik tamamlama, hata düzeltme, kod şablonları gibi özellikler sayesinde yazılım geliştirme süreci daha hızlı ve verimli hale gelir.

2. Entegrasyon: Bir IDE, genellikle farklı araçları ve bileşenleri bir araya getirir. Örneğin, derleme, hata ayıklama, sürüm kontrolü gibi işlevler IDE'nin içinde entegre edilmiştir, bu da geliştiricinin farklı araçlar arasında geçiş yapmasını gerektirmez.

3. Hata Ayıklama Kolaylığı: IDE'ler genellikle kapsamlı hata ayıklama araçları sunar. Değişken değerlerini izleme, adım adım kodu çalıştırma, kesme noktaları belirleme gibi özellikler, hataları bulmayı ve gidermeyi kolaylaştırır.

4. Kod Analizi ve Düzenleme: Birçok IDE, kod kalitesini artırmak için çeşitli analiz araçları sağlar. Kod biçimlendirme, hata tespiti, kod içi belgeleme gibi özellikler geliştiricilere daha sağlam ve okunabilir kodlar yazma imkanı sunar.

5. Platform Bağımsızlık: Birçok IDE, farklı programlama dilleri ve platformlar için destek sunar. Bu da geliştiricilere farklı projelerde tutarlı bir deneyim sağlar.

Bu nedenlerden dolayı, bir IDE kullanmak genellikle yazılım geliştirme sürecini daha etkili hale getirir ve geliştiricilerin daha verimli çalışmalarını sağlar.

**2.** Doğum günün kutlu olsun sözcüklerini içeren bir astring oluşturmak için hangi kodu yazardınız? Yeni oluşturulan nesneye text adlı bir referans göstermek için bir kod yazın.

1 #doğum günün kutlu olsun sözcükleri ile astring oluşturma

2 astring = “Doğum günün kutlu olsun

3 test=astring

4 print(test)

**3.** Oluşturduğunuz dizeyi almak ve iki dizeyi içeren bir listeye bölmek için hangi kodu yazardınız? lst referansını bu yeni oluşturulan listeye yönlendirin.

1 #oluşturulan dize

2 dize=” , “

3

4 #dizeyi ”,” karakterine göre ikiye ayırma

5 ayrik\_dizeler = dize.split (“ , ”)

6

7 #ayrılan dizeyi bir listeye koyma

8 Ist = ayrik\_dizeler

9

10 #sonucu kontrol etmek için listeyi yazdırma

11 print (Ist)

**4.**İkinci sorgularda oluşturulan dizeyi büyük harfle yazmakiçin hangi kodu yazardınız tion.adlı referansı bu büyük harfli dizeye yönlendirin.

UPPER

**5.**Son üç soruyu yanıtlamak için yazdığınız kodu çalıştıracak olsaydınız, Bu üç satır çalıştırıldıktan sonra text tarafından referans verilen dizin ne içerecektir Kod mu?

**6.**"Kent" kelimesini "Denise" kelimesiyle eşleştiren bir sözlüğü nasıl oluşturursunuz? "Steve" mi "Lindy" mi? Bu iki durumda "Kent" ve "Steve" anahtarlardır ve "Denise" ve "Lindy" değerlerdir.

Kent kelimesini denise kelimesine atama işlemi yaparız.

steve kelimesine de lindiyi atarız.

**7.** Bölüm 17'ye bakınız. Sözlük ilk önce boş bir sözlük olarak oluşturulduğunda bir anahtarı bir değere nasıl eşlerdiniz? İPUCU: Bu, ekteki belgelerde "bir öğe ayarlama" olarak adlandırılır. Boş bir sözlük oluşturmak için bir kısa kod parçası yazın ve sonra "Kent"i "Denise" ile ve "Steve"i "Lindy" ile eşleştirin.

CEVAP: Boş bir sözlük oluşturulduğunda, bir anahtarı bir değere eşlemek için "anahtar" kelimesini kullanabilir ve bu işlemi gerçekleştirebilirsiniz. Şu şekilde:

sozluk[anahtar] = deger

Bu, belirtilen anahtara karşılık gelen değeri atanmış olan bir öğe ekler veya var olan bir öğeyi günceller. Örneğin:

sozluk = {}

sozluk["Kent"] = "Denise" sozluk["Steve"] = "Lindy"

Bu kod parçası, "Kent" anahtarını "Denise" değeriyle ve "Steve" anahtarını "Lindy" değeriyle eşler.

**8.** "x < y" ifadesi yazıldığında hangi yöntem çağrılır? Bu yöntem hangi sınıfın bir üyesidir? Başka bir deyişle, bir programda "x < y" ifadesiyle karşılaştığınızda, "x < y" ifadesinin tam olarak ne anlama geldiğini anlamak için hangi sınıfı incelemeniz gerektiğini nasıl bulursunuz?

CEVAP: " x < y " ifadesi yazıldığında çağrılan yöntem \_\_lt\_\_() özel metodu olur. Bu metot, x nesnesinin y nesnesinden küçük olup olmadığını değerlendirir. Bu metot, karşılaştırma operatörlerinin "less than" ( < ) sembolü ile çağrılır.

Bu metodu anlamak için, x nesnesinin sınıfının belgelerini ve metodlarını inceleyebilirsiniz. Eğer x bir özel sınıfın bir örneği ise, bu sınıfın belgelerine bakarak \_\_lt\_\_() metodunu bulabilirsiniz. Eğer x ve y built-in bir tür (örneğin, sayılar) ise, o zaman x'in sınıf belgelerine ve metotlarına bakmak yerine, Python dil belgelerine başvurabilirsiniz. Bu belgeler, Python dilinde bulunan built-in veri tipleri ve bunların nasıl çalıştığı hakkında ayrıntılı bilgi sağlar.

**9.** "x > y" ifadesi yazıldığında hangi yöntem çağrılır?

CEVAP: " x > y " ifadesi yazıldığında, çağrılan metod "\_\_gt\_\_()" dir ve "büyüktür" anlamına gelir. Bu metod, Python'daki karşılaştırma operatörü aşırı yükleme mekanizmasının bir parçasıdır. x'in değerinin y'nin değerinden büyük olup olmadığını değerlendirir.

**10.** Döngü ve yarım problemi nedir ve nasıl çözülür?

CEVAP: Döngü ve yarım problem, bir döngü içinde yarım işlem yapma ihtiyacı olduğunda ortaya çıkar. Örneğin, bir döngüde her ikinci öğeyi işlemek isteyebilirsiniz, ancak her adımda yarım işlem yapmak istiyorsunuz. Bu durum, döngüyü karmaşıklaştırabilir ve istenmeyen sonuçlara neden olabilir. Döngü sürekli olarak tekrarlanır.

Bu problem genellikle bir sayaç değişkeni kullanılarak çözülür. Sayaç, her adımda artırılır ve işlenen öğe sayısını takip eder. Her ikinci adımda, işlem yapılır ve sayaç sıfırlanır. Bu, döngü ve yarım problemi için yaygın bir çözümdür.

**11.**Bir XML dosyasını okumak için döngü ve yarım probleminin çözümünü kullanmanız gerekir mi? Neden ya da neden olmasın?

--> Genellikle XML dosyalarını okurken döngü ve yarım problemi çözümünü kullanmak yerine dilin XML ayrıştırıcılarını kullanmak daha etkili olabilir. XML ayrıştırıcılar, XML dosyalarındaki verilere daha kolay erişim sağlar ve işlemleri daha basit hale getirir. Bu nedenle, genellikle XML dosyalarını okurken döngü ve yarım problemi çözümüne ihtiyaç duyulmaz.

**12.** Polimorfizm ve Operatör Aşırı Yüklemesi birbiriyle yakından ilişkili kavramlardır. İki kavramın nasıl benzer olduğunu ve Python'un bunları nasıl desteklediğini kısaca açıklayabilir misiniz? İPUCU: Hayır geçerli bir cevap değildir.

--> Polimorfizm ve operatör aşırı yüklemesi, nesne yönelimli programlamada kullanılan tekniklerdir. Polimorfizm, farklı sınıfların aynı metot adını kullanabilmesini sağlar. Operatör aşırı yüklemesi ise, operatörlerin farklı veri tipleri üzerinde farklı işlemler yapabilmesini sağlar. Python'da bu teknikleri kullanarak daha esnek ve okunabilir kodlar yazabiliriz.

**13.**Bir programın kullanıcıdan bir tamsayı girmesini istemesi için ne yazarsınız? Daha sonra 2'den itibaren tüm çift tam sayıları kullanıcı tarafından girilen tam sayıya ekler.(toplar) İPUCU:Bunu gerçekleştirmek için aralık fonksiyonunun nasıl kullanılacağını gözden geçirmek isteyebilirsiniz ve hangi programlama modelini kullanacağınıza karar verin.

--> Kullanıcıdan bir tamsayı girmesini istemek için Python programında şu kodları kullanabiliriz:

# Kullanıcıdan tamsayı girişi al

sayi = int(input("Bir tamsayı giriniz: "))

# Toplamı hesaplamak için değişken oluştur

toplam = 0

# 2'den başlayarak girilen tamsayıya kadar olan çift sayıları topla

for i in range(2, sayi+1, 2):

toplam += i

# Sonucu ekrana yazdır

print(f"2'den {sayi}'e kadar olan çift sayıların toplamı: {toplam}")

Bu program kullanıcıdan bir tamsayı alır ve 2'den başlayarak kullanıcının girdiği tamsayıya kadar olan çift sayıları toplar. Sonucu ekrana yazdırır.

**14.** SORU

Tkinter’ı kullanarak nasıl bir pencere oluşrurursunuz?

CEVAP

import tkinter as tk

pencere=tk.Tk()

pencere.mainloop()

**15.** SORU

Tkinter programındaki Frame nesnesinin amacı nedir?

CEVAP

Tkinter öğelerini taşımak ve gruplamak için kullanılır.

16. Tkinter çerçevesindeki üç widget türü nedir?

1. Label (Etiket): Bir metin veya resim içeren etiketler oluşturmak için kullanılır. Kullanıcıya bilgi sağlamak veya arayüzde başlık göstermek için yaygın olarak kullanılır.

2. Button (Düğme): Kullanıcının bir eylemi tetiklemesini sağlamak için kullanılır. Genellikle kullanıcı tarafından tıklanarak belirli bir fonksiyonu çalıştırmak için kullanılır.

3. Entry (Giriş): Kullanıcının metin girmesini sağlayan bir kutu. Genellikle kullanıcıdan veri almak için kullanılır.

17.Bir XML dosyasını okurken, dosyayı okumak için kaç satır kod gerekir?

CEVAP

Bu kod, ornek.xml adlı XML dosyasını okur ve her bir öğe için etiket ve metni ekrana basar. Ancak, kullanılan dil ve kütüphane değiştiğinde, kodun yapısı da değişebilir.

18. Bir XML belgesinden tek bir öğeyi nasıl alırsınız? Hangi kod satır(lar)ı yazmak zorunda mısınız? Bir örnek verin.

CEVAP

Bu örnek kod, belirli bir XPath ifadesini kullanarak XML belgesinden hedeflenen öğeyi alır ve ekrana basar. `./path/to/target\_element` kısmını, belgenizdeki hedef öğenin XPath yoluna göre değiştirmelisiniz. Bu yol belirli bir öğeyi tanımlar ve bu öğenin değerini almanıza olanak tanır.

**20. Bir XML belgesindeki öznitelik nedir ve bir özniteliğin değerine nasıl erişirsiniz? Metinden veya internette bulduğunuz başka bir örnekten bir örnek verin?**

**CEVAP**

XML'deki bir öznitelik, bir XML öğesinin ek, belirli özellikler sağlayan bir parçasıdır. Aşağıdaki biçimde yazılan bir öznitelik adı ve bir öznitelik değerinden oluşur: öznitelik\_adı="öznitelik\_değeri". İşte sağladığınız metinden bir örnek:

<employee id="be129">

<firstname>John</firstname>

<lastname>Doe</lastname>

<title>Engineer</title>

<division>Materials</division>

</employee>

Bu örnekte id öznitelik, "be129" ise değerdir. Nitelikler bir öğe hakkında daha fazla bilgi sağlar ve içindeki içeriği tanımlamaya yardımcı olur.

Başka bir örnek:

<text category="informative" language="en">Hello, World!</text>