Genel Özet

- → *Programcılığın* temelinde *algoritma* geliştirme vardır.
- → *Program*, belirli tasarlanmış/geliştirilmiş kodlardan oluşan bir yazılım parçasıdır.
- → **Veri Yapısı (data structure)**: verilerin saklanma şekli
  - → *verilerin etkin olarak saklanması ve işlenmesi için kullanılan yapı*
  - → *uygun veri yapısı yerden kazandırır, algoritmayı hızlandırır*
- → **Veri Türü**: veriyi açıklamak üzere kurulmuştur
- → **Veri Modeli (data model)**: verilerin ilişkisel düzeni
- → her problem doğası gereği en uygun veri modeline sahiptir
- → veri modeline dayanılarak algoritmik ifadeler ortaya konur

Temel Veri Türleri

→ tamsayı – karakter – gerçel sayı – string vs

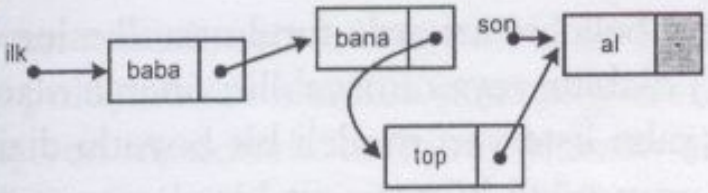
Veri Yapı Sınıfları

- temel
- tanımlamalı
  - *topluluk oluşturma (struct)*
  - *ortaklık oluşturma (union)*
  - *bit düzeyinde erişim*

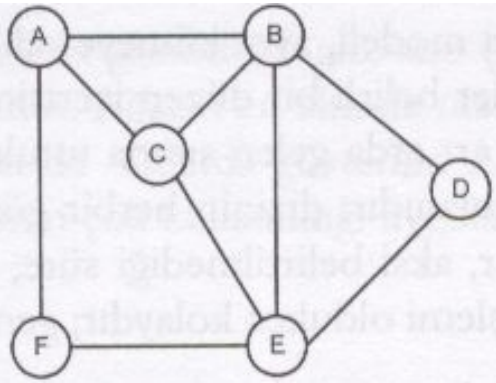
Veri Modelleri

- → bağlantılı liste – ağaç – graf – durum makinesi – veritabanı–ilişkisel – ağ bağlantı
- → hash (çırpı)
- → TODO: şekil eklenecek

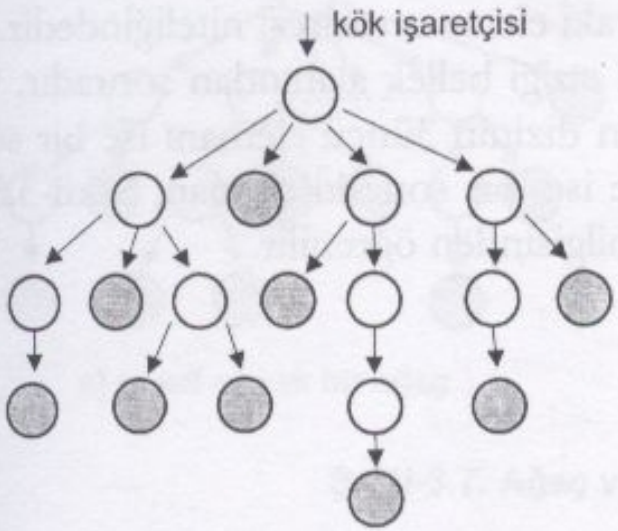
devam



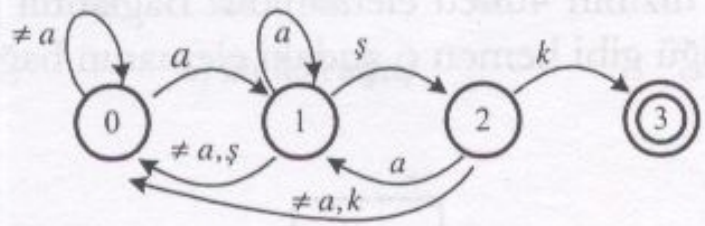
a) bağlantılı liste



c) Graf veri modeli



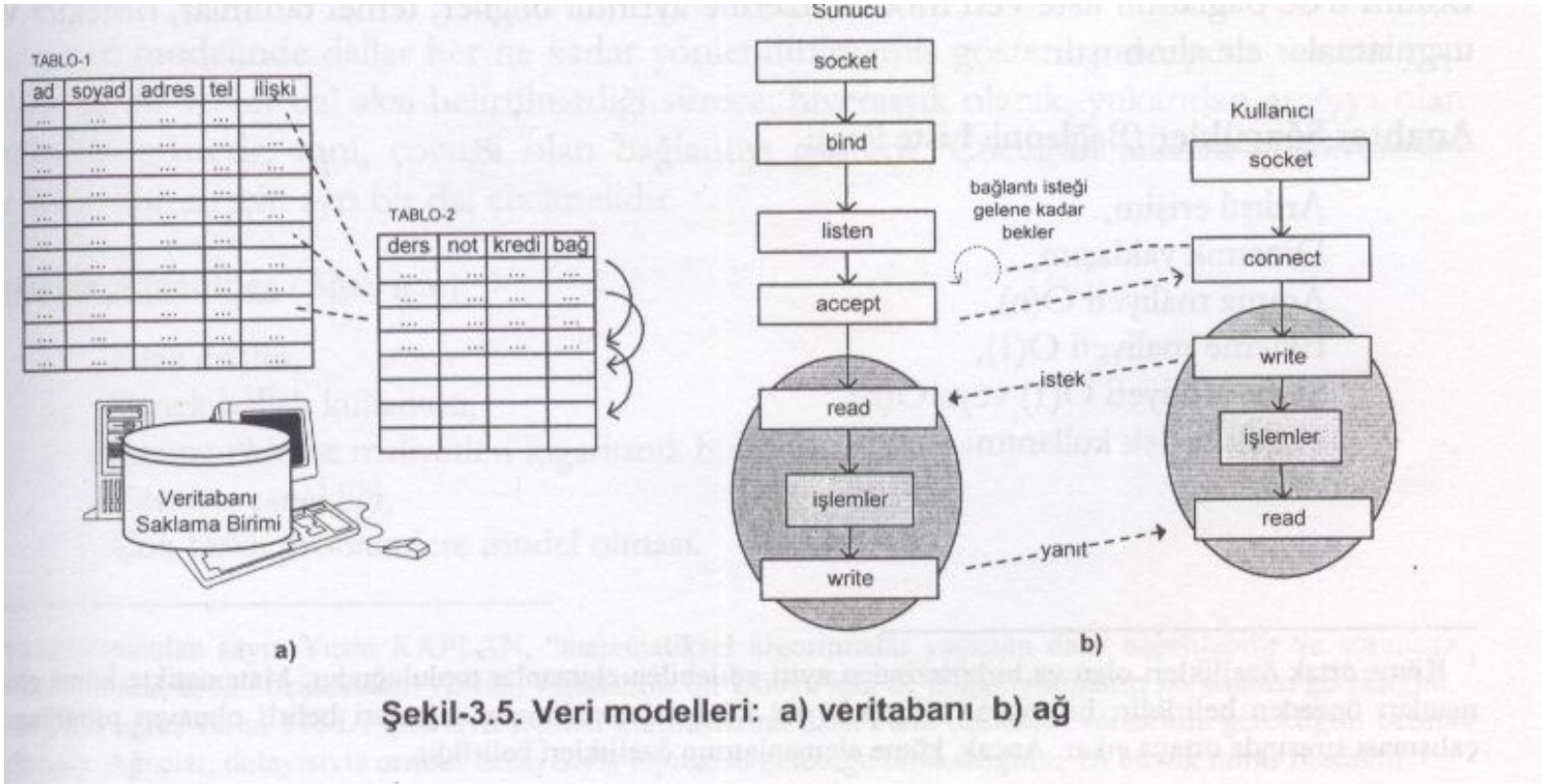
b) Ağaç veri modeli



d) Durum makinası

Şekil-3.4. Çeşitli veri modelleri

devam



Şekil-3.5. Veri modelleri: a) veritabanı b) ağ

Algoritma

→ **Algoritma**: belirli bir işi var olan veya sonradan tanımlanan veri modeline dayandırarak adım adım ortaya koyma

- → **bir sorunun çözümü için öngörülen işlemlerin *mantıksal* ve** *sembolik anlatımı.*
- → *Sedgewick'e göre problem çözme yöntemi*
- → *hayatımızın parçasıdır: işe alınan kişinin oryantasyonu, ...*

→ *Abu Jafar Mohammed ibn Musa el Khowarizmi* (Horazmi)

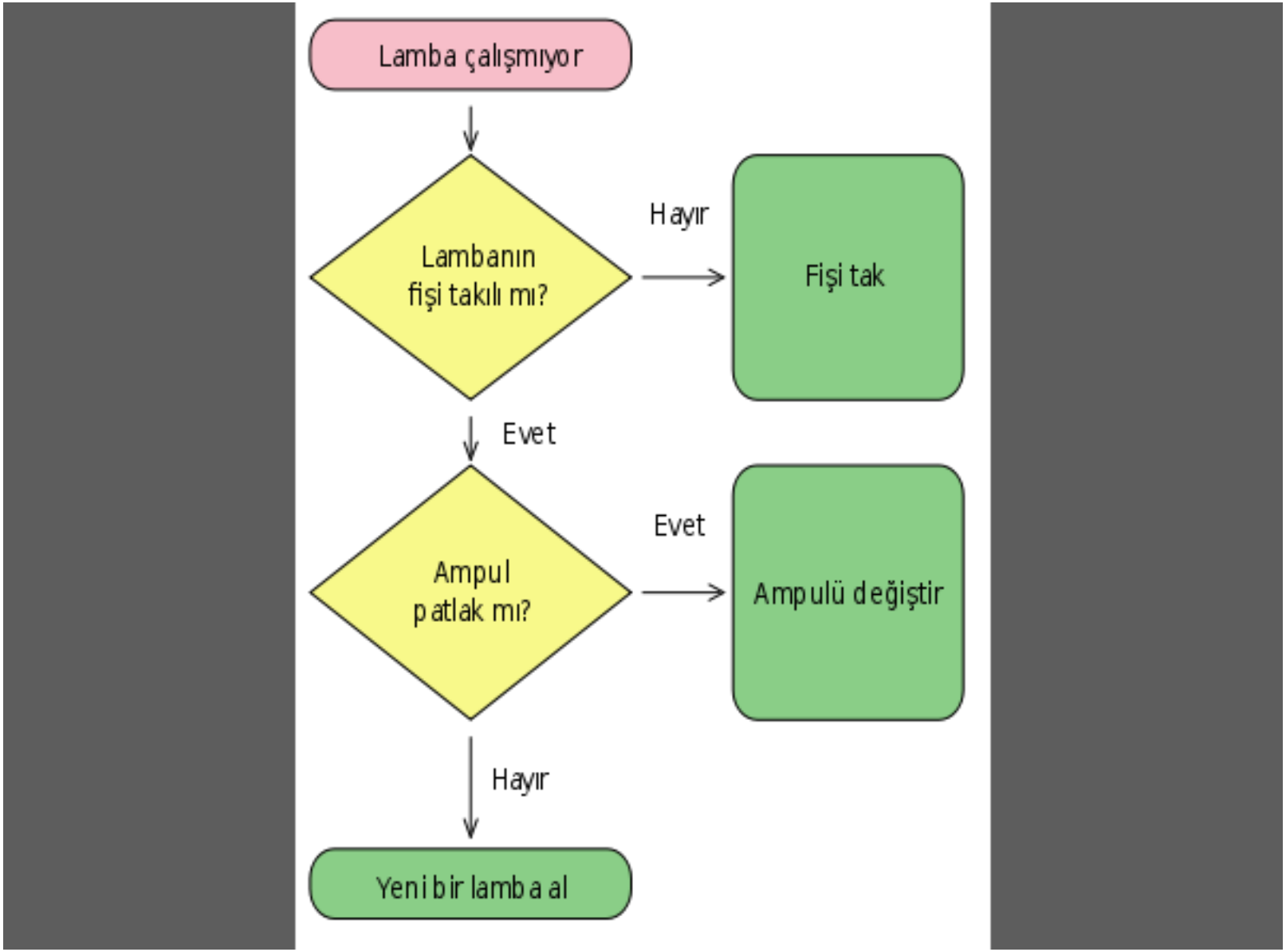
- → *Özbekistan'ın Harezm (Khiva) kenti, Fars bilgini*
- → *"El Horazmi'ye göre ..." deyimi zaman içerisinde "algoritma" kelimesi,*
- → *hesapların anlatımı*
- → *Euclid'in çalışmaları*

→ Sonlu ve düzenli eylemler dizisinin betimlenmesi...Kesin sonuçlu

Algoritmik X Sezsel Düşünme

- → Algoritma: bir sonucun bulunmasını kolaylaştıran, ama deneme-yanılmaya ve sezgisel çözümlemeye karşıt olan bir yöntemdir.
- → Algoritmada çözüm için olası yöntemlerden en uygun olanı seçilir, ve yapılması gerekenler adım adım ortaya konur
- → Sezisel düşünmede ise çözüm açık değildir; tasarımcının deneyimi, birikimi ve o andaki düşüncesine göre problemi çözecek birşeylerin şekillendirilmesiyle yapılır.
- → Bazı problemler (network cennectivity) ancak algoritmik düşünceyle çözülebilir.
- → Algoritmanın özel geometrik şekillerle çizilmiş hali: **akış diyagramı**, N-S ve W-O şemaları

Akış Çizgesi



Algoritmalarla kimler çalışır

- → Internet. Web search, packet routing, distributed file sharing.
- → Biology. Human genome project, protein folding.
- → Computers. Circuit layout, file system, compilers.
- → Computer graphics. Movies, video games, virtual reality.
- → Security. Cell phones, e-commerce, voting machines.
- → Multimedia. CD player, DVD, MP3, JPG, DivX, HDTV.
- → Transportation. Airline crew scheduling, map routing.
- → Physics. N-body simulation, particle collision simulation.

Algoritmalar

- → 20.yy bilimi formüle dayalı
- → 21.yy bilimi algoritmaya dayalı
- → algoritma: doğanın - insanın - bilgisayarın ortak dili (Avi Wigderson)
- → Bir çok algoritmaya ihtiyaç duyulur
  - → *Arama - Sıralama - Matrissel - Graf- Matematiksel Problem*

Algoritmanın Özellikleri

- → Etkinlik: tekrarların olmaması, genel amaçlılık
- → Sonluluk: özyinelilik
- → Kesinlik: kesin sonuç/netice, kesin sıralılık (adımların sırası)
- → sırayı belirleyen kumanda yapısı
- → girdi/çıktı olmalıdır
- → başarım

Hesap, Programlama Dilleri ve Algoritmalar

- → Gündelik yaşamımızda her zaman hesaplarız: paramızı, puanımızı, değerlerimizi, yaşımızı vb
- → nitel->nicel: ölç->hesap
- → bir algoritmanın gerçekleştirilmesi için programlanması gerekir -> programlama dili.

Soyutlama

Soyutlama: problem ve çözümü mantıksal ve fiziksel yönleriyle ayırarak görmeye imkan verir.

→ Otomobil. Sürücü (kullanıcı) ve Tamirci arasındaki fark. Mantıksal ve fiziksel bakış

→ kullanıcıya sunulan hizmetler **arayüz** olarak adlanır.

→ tamirci "kaputun altındaki" ayrıntılarla da ilgilenir.

→ bilgisayar kullanıcıları için de geçerli. Mantıksal veya kullanıcı penceresinden görüyorlar.

→ Bilg.Bil., teknoloji destek elemanları, programcılar ve sistem yöneticileri, alt-seviye ayrıntıları denetlerler.

→ Soyutlamanın kullananı = **istemci**, işler yolunda gittiği sürece arayüz doğru yanıt verdiği sürece

→ ayrıntıları bilmesine gerek yoktur.

İşlevsel Soyutlama

→ Python:Math modülü

```
>>> import math
>>> math.sqrt(16)
4.0
>>>
```

Kara Kutu

→ bir arayüz tanımlarız, neye gerek duyuyor ve ne döndürecek, ve ayrıntılar içine gömülür

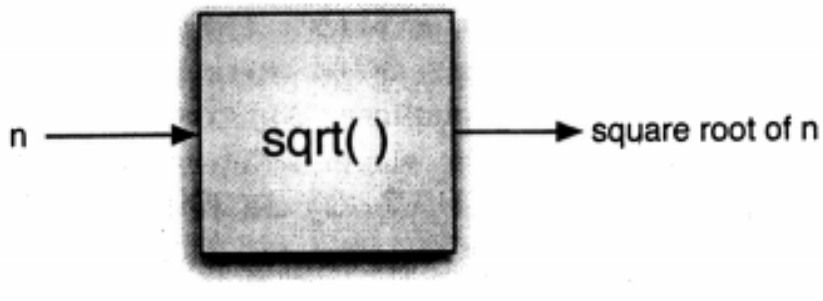


Figure 1.1: Procedural Abstraction

1.3.1 Programlama Nedir?

Programlama: bir algoritmaya göre bilgisayarın işleyebileceği/yürütebileceği emirlere notasyona (programlama diline) dönüştürme sürecidir.

→ Bilg.Bil.'de programlama önemli bir araç olmanın ötesi değildir.

→ Programlama, problem çözümüne temsil üretmek

→ Programlama dili, algoritmanın temsil edilmesi

Veri Türü

- → bilgisayarda her şey sayı: 1/0, ikil katar
- → bu katarlara anlam veri türleriyle sağlanır
- → çözmeye çalıştığımız probleme ait veri hakkında, ikil veriden yorumlama imkanı verir.

Örnek: Veri Türü: Tamsayı

- → veri: ikil bit katarı --> yorum: tamsayı
- → örn. 23, 654, -19
- → hangi işlemler yürütülebilir: toplama, çıkartma, çarpma vs.

Karmaşık Veri Türleri

→ problem ve çözümü aşırı derecede zor olduğunda,

→ bu basit veri türleri ve denetim yapıları problem çözme sürecinde zorluklara neden olur.

→ Bu karmaşıklığı idare etmenin ve çözümü üretmenin kolay bir yolu olmalı!

1.3.2 Veri Türleri ve Soyut Veri Türleri

- → problemin ve problem-çözmenin karmaşıklığını idare etmek: soyutlama gereksinimi
- → "büyük resme" odaklanmak
- → problem düzleminde modeller üretmek
- → model: problemin doğasına uygun, idaresi kolay, daha verimli

Veri Soyutlama

Soyut Veri Türü (ADT): nasıl tasarlandığına dair ayrıntıları göz ardı ettirecek veri parçaları ve işlemler için mantıksal tanımlama düzlemi.

$\longrightarrow$ tasarım ayrıntıları yerine kullanımına odaklanma imkanı

Veri Sarmalama

→ Veri sarmalamayla gerçeklemeye dair ayrıntıları kullanıcıdan gizleriz

→ bir anlamda bilgi gizlemedir

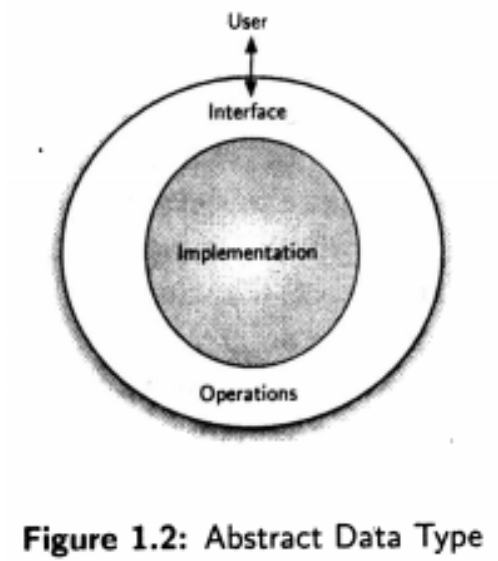


Figure 1.2: Abstract Data Type

1.3.3 Algoritmalarla Neden Çalışırız?

- → deneyimle öğrenme
- → diğerlerinin çözümlerini görüp kendimiz çözerek öğreniriz
- → benzer problemlerde belli örüntüleri yakalayıp, hızlı ve şık tasarımlar yapmamız mümkün olur

- → Python, Nesne Yönelimlidir (*OOP*)
- → betik dilidir
- → prompt >>>
- → satır-satır yorumlar ve sonuçları gösterir

```
>>> print "Merhaba"
Merhaba
>>>
```

1.4.1 Veri

- → Python = OOP
- → verinin görünüşünü (*durum*) ve yapabileceklerini (*davranış*) tanımlamak için **sınıf (class)** tanımlarız.
- → OOP'da veri öğeleri == **nesneler (objects)**
- → bir nesne, bir sınıfın **örneğidir (instance)**

1.4.1.1 İlkel Sınıflar

- → üç yerleşik sayısal sınıf: tamsayı, uzun tamsayı ve kayar noktalı.
- → standart aritmetik işlemler: +, -, * ve ** (üst alma) ve parantezle öncelik ayarı
- → % modül işleci.
- → bu gibi işlemler **yöntem (method)**

devam

```
>>> 2+3*4
14
>>> (2+3)*4
20
>>> 2**10
1024
>>> 6/3
2
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
>>> 7%3
1
>>> 3/6
0
>>> 3.0/6
0.5
>>> 2**100
1267650600228229401496703205376L
```

Mantık Sınıfı

→ Doğru (True) ve Yanlış (False)

→ mantıksal işlemler: and, or, not gibi

```
>>> True
True
>>> False
False
>>> False or True
True
>>> not (False or True)
False
>>> True and True
True
>>>
```

devam

→ Eşitlik (==) ve büyüktür (>) gibi karşılaştırma işlemlerinin sonuçları olarakta karşımıza çıkar

```
>>> 5==10
False
>>> 10 > 5
True
>>>
```

Tanımlayıcılar

- → isimler
- → bir harf veya altçizgiyle (_) başlar
- → büyük/küçük harf duyarlıdır
- → herhangi bir uzunlukta olabilir
- → kod okunurluğu ve anlaşılırlığı yüksek isimler tercih edin!
- → *değişkenler*, atama ifadesinin sol yanında yer aldığı ilk yerde oluşturulur
- → değişken, veri parçasına bir *referanstır*, verinin kendine değil!

devam

→ kod

```
>>> sum = 0
>>> sum
0
>>> sum = sum + 1
>>> sum
1
>>> sum = True
>>> sum
True
```

devam

→ sum = 0 ifadesi sum isminde bir değişken ve 0 veri nesnesine bir *referans* oluşturur.

Figure 1.3: Variables Hold References to Data Objects

→ genelde atama ifadesinin sağ yanı hesaplanır, elde edilen sonuca bir referans değişkene "atanır"

devam

→ verinin türü değiştiğinde, Python dinamik karakteristiğe sahiptir

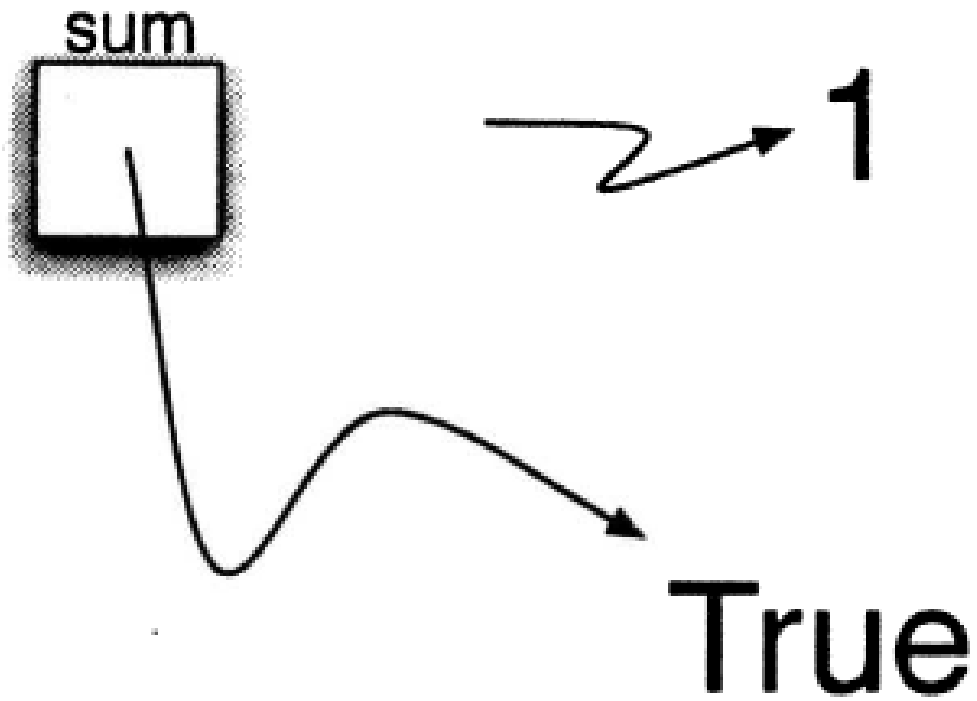


Figure 1.4: Assignment Changes the Reference

1.4.1.2 Yerleşik Kolleksiyonlar: Liste

- → sıfır veya daha fazla sayıda Python veri nesnesini sıralı tutar
- → öğeleri virgülle (,) ve köşeli parantezle ([ ]) hapsedilir
- → boş liste: [ ].
- → listeler heterojendir, elemanları aynı türden olmak zorunda değildir!

```
>>> [1, 3, True, 6.5]
[1, 3, True, 6.5]
>>> mylist = [1, 3, True, 6.5]
>>> mylist
[1, 3, True, 6.5]
>>>
```

Ardışıllık üzerinde desteklenen işlemler

Operation Name	Operator	Explanation
Indexing	[ ]	Access an element of a sequence
Concatenation	+	Combine sequences together
Repetition	*	Concatenate a repeated number of times
Membership	in	Ask whether an item is in a sequence
Length	len	Ask the number of items in the sequence
Slicing	[:]	Extract a part of a sequence

Table 1.1: Operations on Any Sequence in Python

devam

```
>>> mylist
[1, 3, True, 6.5]
>>> mylist[2]
True
>>> mylist + mylist
[1, 3, True, 6.5, 1, 3, True, 6.5]
>>> False in mylist
False
>>> mylist * 3
[1, 3, True, 6.5, 1, 3, True, 6.5, 1, 3, True, 6.5]
>>> mylist[1:3]
[3, True]
>>> len(mylist)
4
>>> len(mylist*4)
16
>>>
```

devam

- → liste indisleri 0 ile başlar
- → dilimleme işlemi mylist[1:3], 1'den başlar, 3 dahil olmamak üzere parçayı alır
- → Tekrarlama işlemi veriden ziyade referansın tekrarıdır!

devam

→ kod

```
>>> mylist = [1,2,3,4]
>>> A = [mylist]*3
>>> A
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
>>> mylist[2]=45
>>> A
[[1, 2, 45, 4], [1, 2, 45, 4], [1, 2, 45, 4]]
>>>
```

→ A değişkeni mylist isimlenen değişkene üç referans içerir

→ bu yüzden mylist'teki bir değişiklik tüm referanslarda etki gösterir.

range işlevi

→ tamsayı listesi oluştururken range işlevine başvururuz

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1,10,3)
[1, 4, 7]
>>>
```

Tekrarlama (*) İşlemi

→ Listeleri ilklerken tekrarlama işleminden yararlanabiliriz

```
>>> mylist = [23] * 6
>>> mylist
[23, 23, 23, 23, 23, 23]
>>>
```

Liste Yöntemleri

Method Name	Use	Explanation
append	`alist.append(item)`	Adds a new item to the end of a list
insert	`alist.insert(i,item)`	Inserts an item at the ith position in a list
pop	`alist.pop()`	Removes and returns the last item in a list
pop	`alist.pop(i)`	Removes and returns the ith item in a list
sort	`alist.sort()`	Modifies a list to be sorted
reverse	`alist.reverse()`	Modifies a list to be in reverse order
del	`del alist[i]`	Deletes the item in the ith position
index	`alist.index(item)`	Returns the index of the first occurrence of `item`
count	`alist.count(item)`	Returns the number of occurrences of `item`
remove	`alist.remove(item)`	Removes the first occurrence of `item`

Table 1.2: Methods Provided by Lists in Python

devam

```
>>> mylist
[1024, 3, True, 6.5]
>>> mylist.append(False)
>>> mylist
[1024, 3, True, 6.5, False]
>>> mylist.insert(2,4.5)
>>> mylist
[1024, 3, 4.5, True, 6.5, False]
>>> mylist.pop()
False
>>> mylist
[1024, 3, 4.5, True, 6.5]
>>> mylist.pop(1)
3
>>> mylist
[1024, 4.5, True, 6.5]
```

devam

```
>>> mylist.pop(2)
True
>>> mylist
[1024, 4.5, 6.5]
>>> mylist.sort()
>>> mylist
[4.5, 6.5, 1024]
>>> mylist.reverse()
>>> mylist
[1024, 6.5, 4.5]
>>> mylist.count(6.5)
1
>>> mylist.index(4.5)
2
>>> mylist.remove(6.5)
>>> mylist
[1024, 4.5]
>>> del mylist[0]
>>> mylist
[4.5]
```

Basit veri nesne yöntemleri

→ kod

```
>>> (54).__add__(21)
75
>>>
```

→ tabii ki basit-kısa olarak 54 + 21

Stringler (karakter katarları)

→ yalnızca karakter içeren listedir

→ tek (') veya çift tırnak (") kullanılabilir

```
>>> "Nurettin"
'Nurettin'
>>> myname = "Nurettin"
>>> myname[3]
'e'
>>> myname*2
'NurettinNurettin'
>>> len(myname)
8
>>>
```

Stringler için sağlanan işlevler

Method Name	Use	Explanation
`center`	`astring.center(w)`	Returns a string centered in a field of size `w`
`count`	`astring.count(item)`	Returns the number of occurrences of `item` in the string
`ljust`	`astring.ljust(w)`	Returns a string left-justified in a field of size `w`
`lower`	`astring.lower()`	Returns a string in all lowercase
`rjust`	`astring.rjust(w)`	Returns a string right-justified in a field of size `w`
`find`	`astring.find(item)`	Returns the index of the first occurrence of `item`
`split`	`astring.split(schar)`	Splits a string into substrings at `schar`

Table 1.3: Methods Provided by Strings in Python

devam

```
>>> myname
'Nurettin'
>>> myname.upper()
'NURETTIN'
>>> myname.center(20)
'      Nurettin      '
>>> myname.find('e')
3
>>> myname.split('e')
['Nur', 'ttin']
>>>
```

Değiştirilebilirlik (*mutuability*)

→ kod

```
>>> mylist
[1, 3, True, 6.5]
>>> mylist[0] = 2**10
>>> mylist
[1024, 3, True, 6.5]
>>>
>>> myname
'Nurettin'
>>> myname[0] = 'X'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

→ Listeyle String arasındaki en büyük fark *değiştirilebilirliktir*

Tuple

→ Tuple'lar listelere benzerdir, heterojendir

→ string gibi değiştirilemezdir

→ öğeleri virgülle (,) ayrılır, parantezle (( )) hapsedilir

```
>>> mytuple = (2, True, 4.96)
>>> mytuple
(2, True, 4.96)
>>> len(mytuple)
3
>>> mytuple[0]
2
>>> mytuple * 3
(2, True, 4.96, 2, True, 4.96, 2, True, 4.96)
>>> mytuple[0:2]
(2, True)
>>>
```

Tuple: değiştirilebilirlik?

→ tuple'daki bir öğeyi değiştirmeye çalıştığınızda hata mesajı alırsınız

```
>>> mytuple[1] = False
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

Sözlükler

→ key:value çiftleri biçiminde veri tutulur

→ öğeleri virgülle (,) ayrılır, kıvrık parantezle ({ }) hapsedilir

```
>>> capitals = {'Turkiye':'Ankara'}
>>> capitals = {'Turkiye':'Ankara', 'Yunanistan':'Atina',
    ... 'Irak':'Bagdat'}
>>> capitals
{'Yunanistan': 'Atina', 'Turkiye': 'Ankara', 'Irak': 'Bagdat'}
>>>
```

Sözlükler: key ile erişim

→ indis yerine key ile erişim, değiştir, ekle, sil.

→ ilk eklenen ilk sırada olmayabilir!

```
>>> capitals['Azerbaycan'] = 'Baku'
>>> capitals
{'Azerbaycan': 'Baku', 'Yunanistan': 'Atina', 'Turkiye': 'Anka
    ... 'Irak': 'Bagdat'}
>>> len(capitals)
4
>>>
```

Sözlükler için sağlanan işlevler

→ image:: tablo14.png

```
>>> phoneext = {'david':1410, 'brad':1137}
>>> phoneext
{'brad': 1137, 'david': 1410}
>>> phoneext.keys()
['brad', 'david']
>>> phoneext.values()
[1137, 1410]
>>> phoneext.items()
[('brad', 1137), ('david', 1410)]
>>> phoneext.get('brad')
1137
>>> phoneext.get('kent')
>>> phoneext.get('kent', 'NO ENTRY')
'NO ENTRY'
>>>
```

1.4.2 Denetim Yapıları: while

→ kod

```
>>> counter = 1
>>> while counter <= 5:
...     print "Hello, world!"
...     counter = counter + 1
...
Hello, world !
Hello, world !
Hello, world !
Hello, world !
Hello, world !
>>>
```

→ koşul kısmında bileşik yapılar kullanılabilir

→ while counter <= 5 and not done:

for

```
>>> for item in [1,3,6,2,5]:
...     print item
...
1
3
6
2
5
>>>
```

for + range

```
>>> for item in range(5):
...     print item**2
...
0
1
4
9
16
>>>
```

for: Stringdeki her bir karakter

→ kod

```
>>> wordlist = ['cat', 'dog', 'rabbit']
>>> letterlist = []
>>> for aword in wordlist:
...     for aletter in aword:
...             letterlist.append(aletter)
...
>>> letterlist
['c', 'a', 't', 'd', 'o', 'g', 'r', 'a', 'b', 'b', 'i', 't']
>>>
```

if: else

→ kod

```
if score >= 90:
    print 'A'
else:
    if score >= 80:
    print 'B'
    else:
    if score >= 60:
        print 'D'
    else:
        print 'F'
```

if: elif

→ kod

```
if score >= 90:
    print 'A'
elif score >= 80:
    print 'B'
elif score >= 60:
    print 'D'
else:
    print 'F'
```

if: tek başına

→ kod

```
if n < 0:
    n = abs(n)
print math.sqrt(n)
```

List Comprehension

→ kod

```
>>> sqlist = []
>>> for x in range(1,11):
...     sqlist.append(x*x)
...
>>> sqlist
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

List Comprehension - 2

→ kod

```
>>> sqlist = [x*x for x in range(1,11)]
>>> sqlist
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
>>>
>>> sqlist = [x*x for x in range(1,11) if x%2 != 0]
>>> sqlist
[1, 9, 25, 49, 81]
>>>
>>>
>>> [ch.upper() for ch in 'comprehension' if ch not in 'aeiou'
['C', 'M', 'P', 'R', 'H', 'N', 'S', 'N']
>>>
```

1.4.3 İşlev Tanımlama

→ kod

```
>>> def square(n):
...     return n**2
...
>>> square(4)
16
>>> square(square(3))
81
>>>
```

Karekök alma: Newton Yöntemi

→ kod

```
def squareroot(n):
    root = n/2
    for k in range(20):
        root = (1.0/2)*(root + (n / root))
    return root
```

1.4.4 OOP: Sınıf Tanımlama

→ Python OOP bir dildir.

→ Yeni sınıflar üretilebilir

→ Veri nesnesinin nasıl göründüğünü (durumu) ve

→ ne yapabileceğini (yöntemler)

→ mantıksal olarak tanımlamak için

→ ADT (soyut veri türü) kullanılır.

→ Soyut veri türü ise sınıf yardımıyla sağlanır,

→ bu durumda programcıya soyutlamanın üstünlükleri de sağlanır.

1.4.4.1 Kesir Sınıfı

→ Kesir iki parçadan oluşur: pay ve payda.

→ Kesrin pay ve paydası tamsayı.

```
class Fraction:

    #yontemler buraya
```

→ Tüm sınıflar kurucu (constructor) yöntemi vardır

→ Python'da __init__

devam

```
class Fraction:

    def __init__(self,top,bottom):

        self.num = top
        self.den = bottom
```

devam

- → üç öğe içerir: self, top, bottom.
- → self: nesnenin kendisine referans. İlk parametre olmalı, çağırırken verilmez.
- → self.num: payı, self.den: paydayı.
- → kurucu yöntemde ilklendirme yapılmıştır.
- → Kesir sınıfından bir örnek (instance) oluşturmak için,

```
myFraction = Fraction(3, 5)
```

- → bu 3/5 değerli myFraction kesir nesnesini oluşturur.

devam

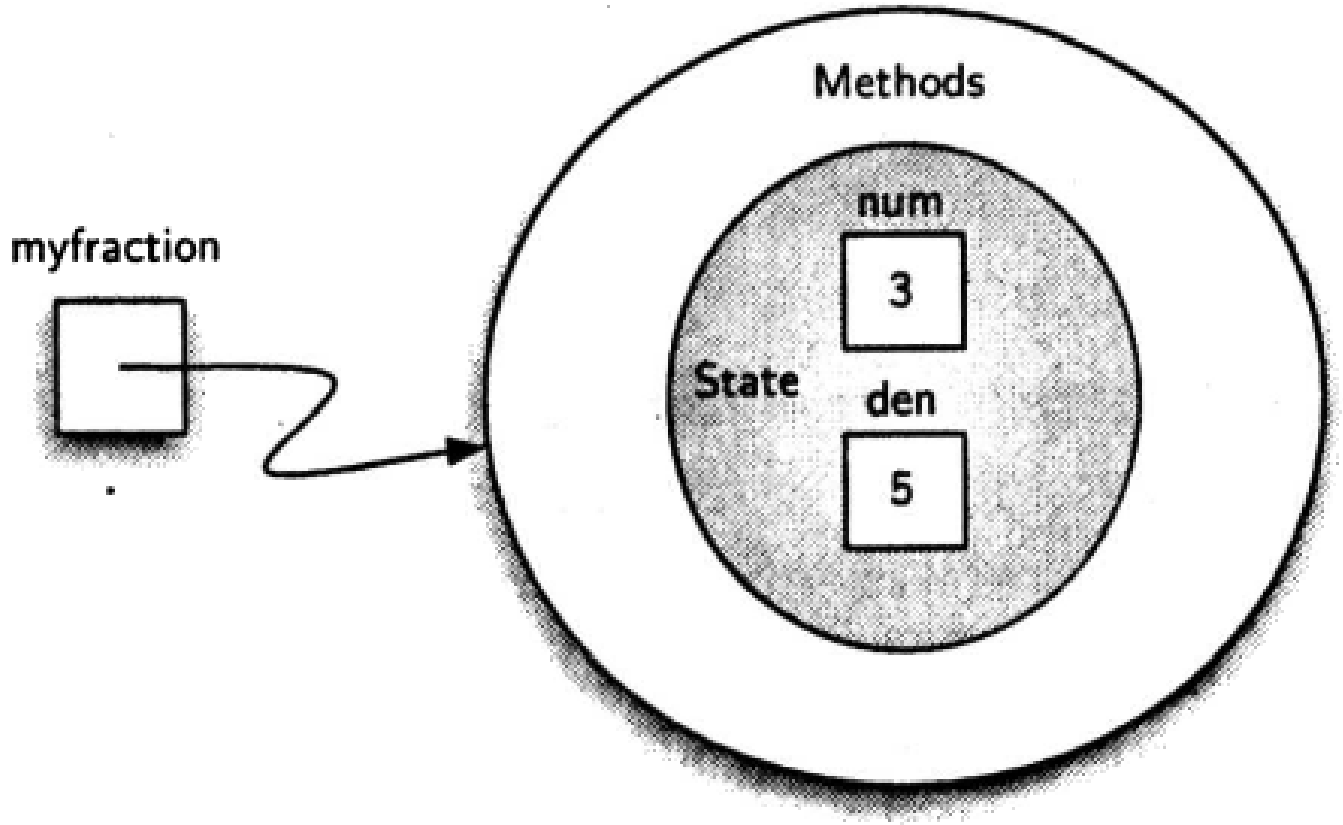


Figure 1.5: An Instance of the `Fraction` Class

→ Kesir sınıfının gerek duyduğu yetenekleri kazandırmak için

devam

→ örn. ekrana yazdırmak

```
>>> myf = Fraction(3, 5)
>>> print myf
<__main__.Fraction instance at 0xb7d92a8c>
>>>
```

→ kesir nesnesi myf, print işleviyle çağrıldığında nasıl tepki vereceğini bilmiyor.

→ bu yüzden print myf bildiği tek şey olan bellek adresini ekrana döküyor.

devam

→ Bu sorunu çözmenin iki yolu var

1. show isimli yöntem yazmak
2. __str__ yöntemini güncellemek

→ ilk yöntem genel olmamakla birlikte

```
def show(self):
    print self.num,"/",self.den
```

devam

```
>>> myf = Fraction(3, 5)
>>> myf.show()
3 / 5
>>> print myf
<__main__.Fraction instance at 0x95c088c>
>>>
```

→ ikincisi daha geneldir. Tüm sınıflara sağlanan yerleşik __str__ işlevinden yararlanılır.

→ ki bu işlev nesneyi stringe çevirir.

__str__ yöntemi: üzerine Bindirme (Override)

→ Python'un sunduğu yerleşik __str__ yöntemini kullan

```
def __str__(self):
    return str(self.num)+"/"+str(self.den)
```

```
>>> myf = Fraction(3, 5)
>>> print myf
3/5
>>>
```

Aritmetik destek

```
>>> f1 = Fraction(1,4)
>>> f2 = Fraction(1,2)
>>> f1+f2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'instance' and 'instance'
>>>
```

__add__ yöntemi

→ Tasarım

```
def __add__(self,otherfraction):

    newnum = self.num*otherfraction.den + \
             self.den*otherfraction.num
    newden = self.den * otherfraction.den

    return Fraction(newnum,newden)
```

→ Test

```
>>> f1 = Fraction(1,4)
>>> f2 = Fraction(1,2)
>>> f3 = f1.__add__(f2)
>>> print f3
6/8
>>> f3 = f1 + f2
>>> print f3
6/8
>>>
```

Sonuçların iyileştirilmesi: GCD

→ 6/8 yerine 3/4

→ Euclid'in GCD algoritması

```
#Assume that m and n are greater than zero
def gcd(m,n):
    while m%n != 0:
        oldm = m
        oldn = n

        m = oldn
        n = oldm%oldn

    return n
```

devam

```
>>> f1 = Fraction(1,4)
>>> f2 = Fraction(1,2)
>>> f3 = f1.__add__(f2)
>>> f3 = f1 + f2
>>> print f3
3/4
>>>
```

Son durum

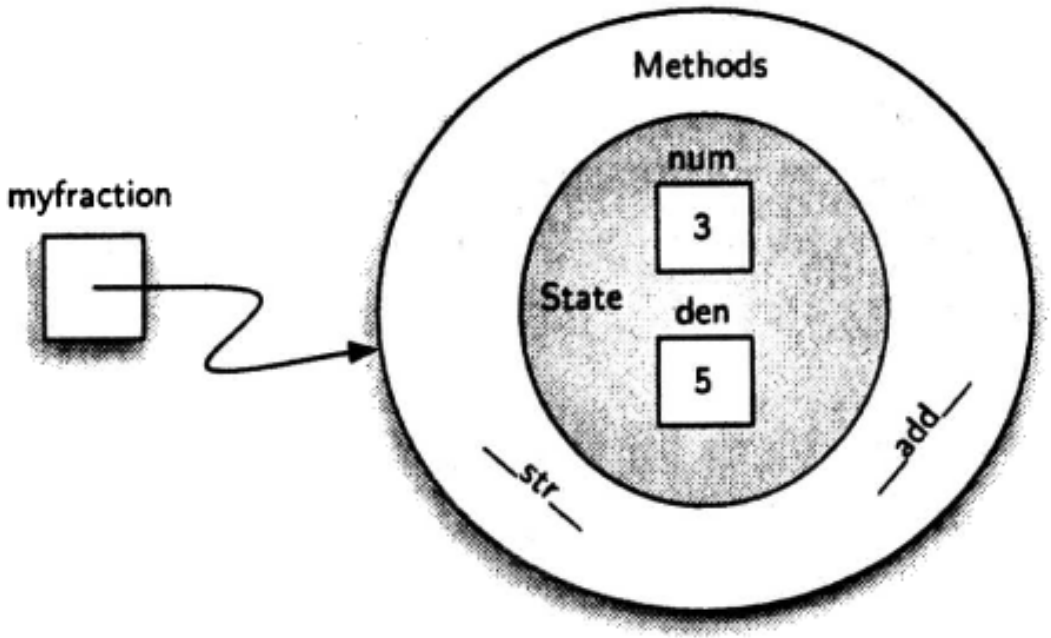


Figure 1.6: An Instance of the `Fraction` Class with Two Methods

Shallow Equality x Deep Equality

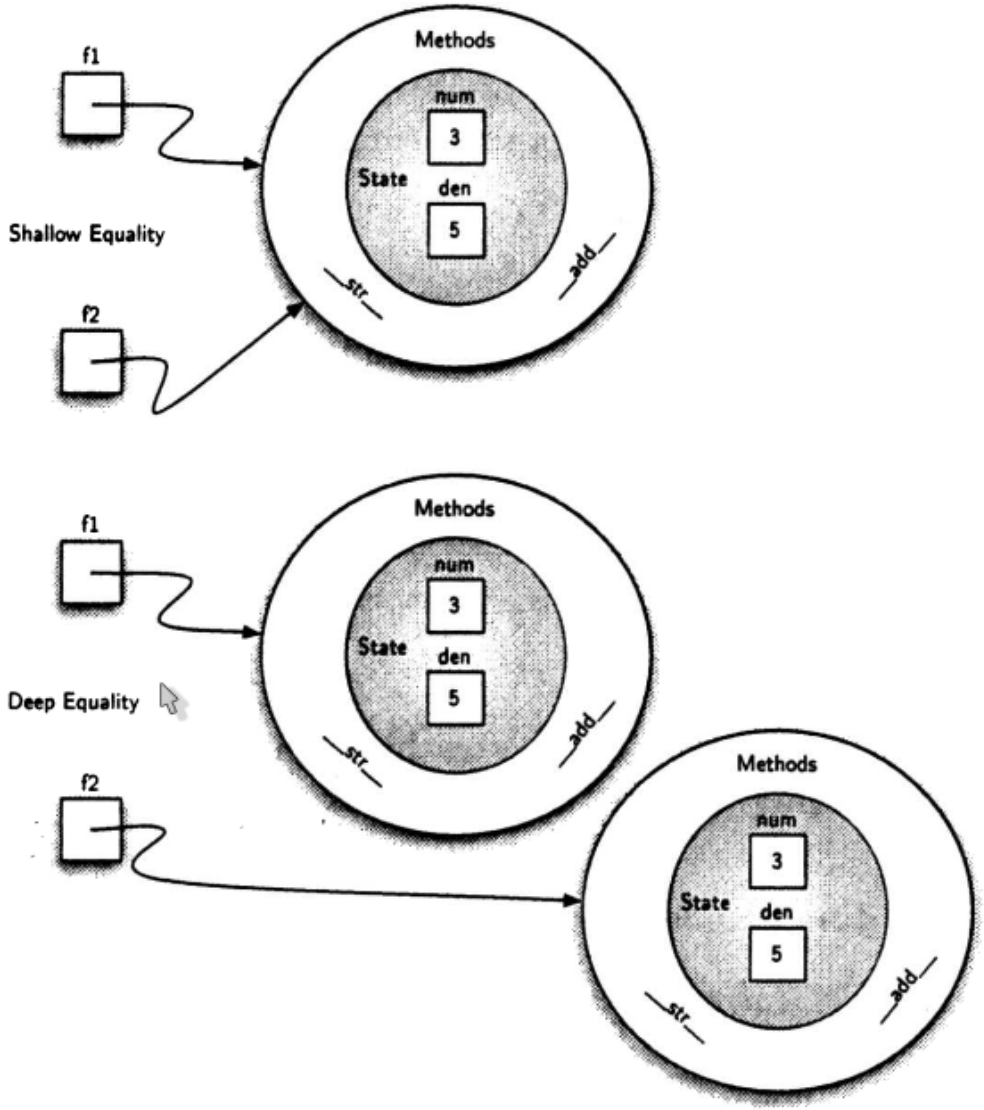


Figure 1.7: Shallow Equality Versus Deep Equality

Karşılaştırma nasıl: __cmp__

→ f1 == f2 x bellek adresi karşılaştırma

```
def __cmp__(self,otherfraction):

    num1 = self.num*otherfraction.den
    num2 = self.den*otherfraction.num

    if num1 < num2:
        return -1
    else:
        if num1 == num2:
            return 0
        else
            return 1
```

Fraction sınıfı

```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num)+"/"+str(self.den)

    def show(self):
        print self.num,"/",self.den

    def __add__(self,otherfraction):
        newnum = self.num*otherfraction.den + \
                 self.den*otherfraction.num
        newden = self.den * otherfraction.den
        common = gcd(newnum,newden)
        return Fraction(newnum/common,newden/common)

    def __cmp__(self,otherfraction):
        num1 = self.num*otherfraction.den
        num2 = self.den*otherfraction.num
        if num1 < num2:
            return -1
          else:
            if num1 == num2:
                return 0
```

1.4.4.2 Miras: Mantık Kapıları ve Devreler

Miras (Inheritance) bir sınıfı başka bir sınıfla ilişkilendirme yöntemi.

→ ebeveyn (*parent*) - çocuk - miras

→ altsınıf - süpersınıf

Python Miras Hiyerarşisi

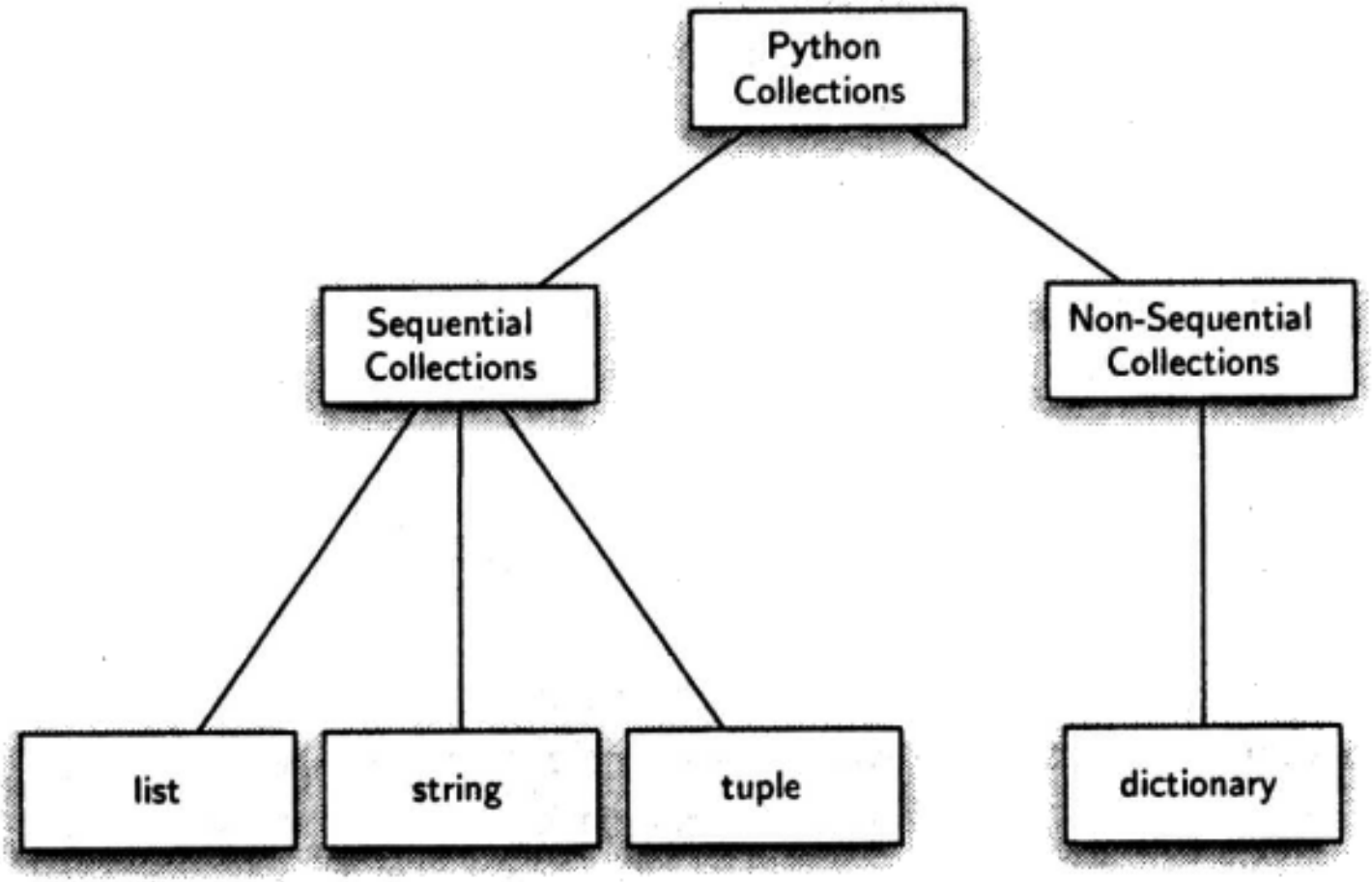


Figure 1.8: An Inheritance Hierarchy for Python Collections

devam

→ liste, ardışıl koleksiyonu nun çocuğudur

→ liste: altsınıf, ardışıl koleksiyonu: süpersınıf

→ list **IS-A** sequantial collection.

IS-A "D is-a B", D sınıfı B sınıfının altsınıfı.

Mantık Kapıları ve Doğruluk Tabloları

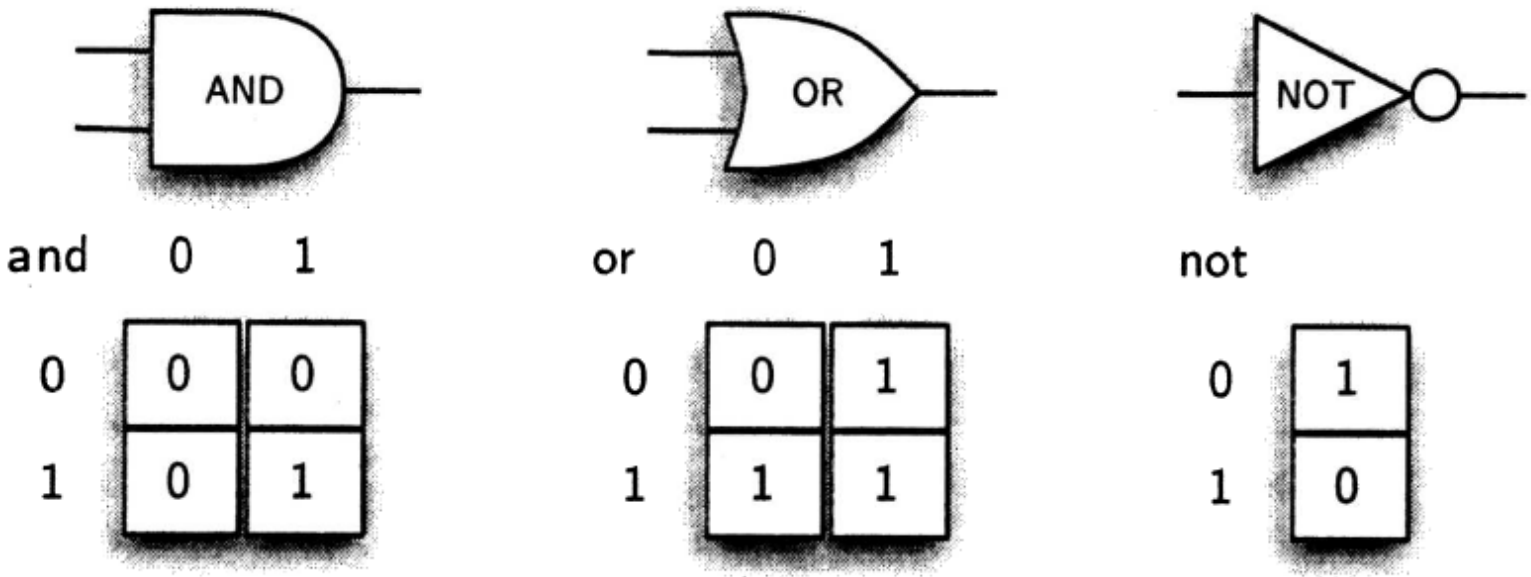


Figure 1.9: Three Types of Logic Gates

Devre

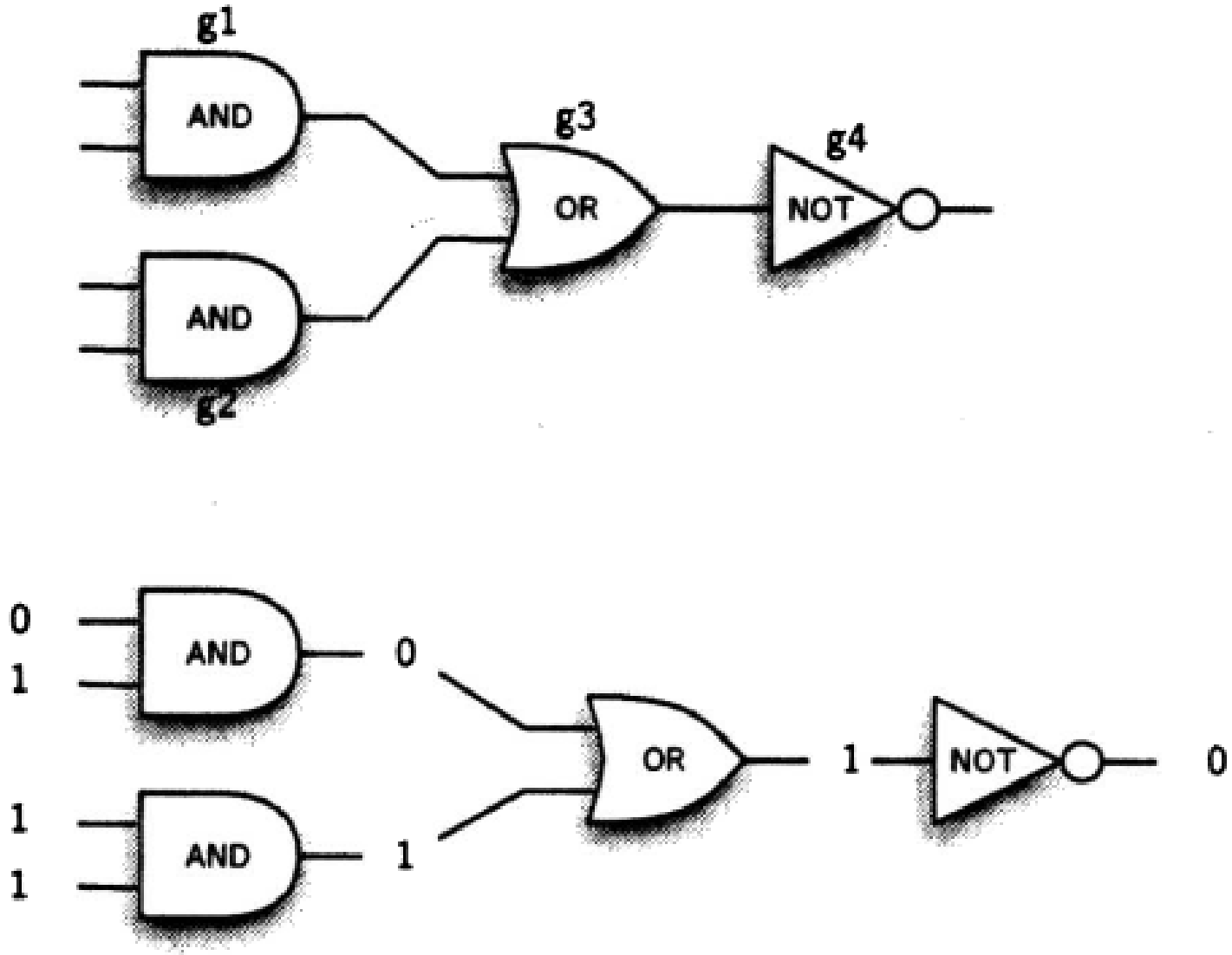


Figure 1.10: Circuit

Mantık Kapı Hiyerarşisi

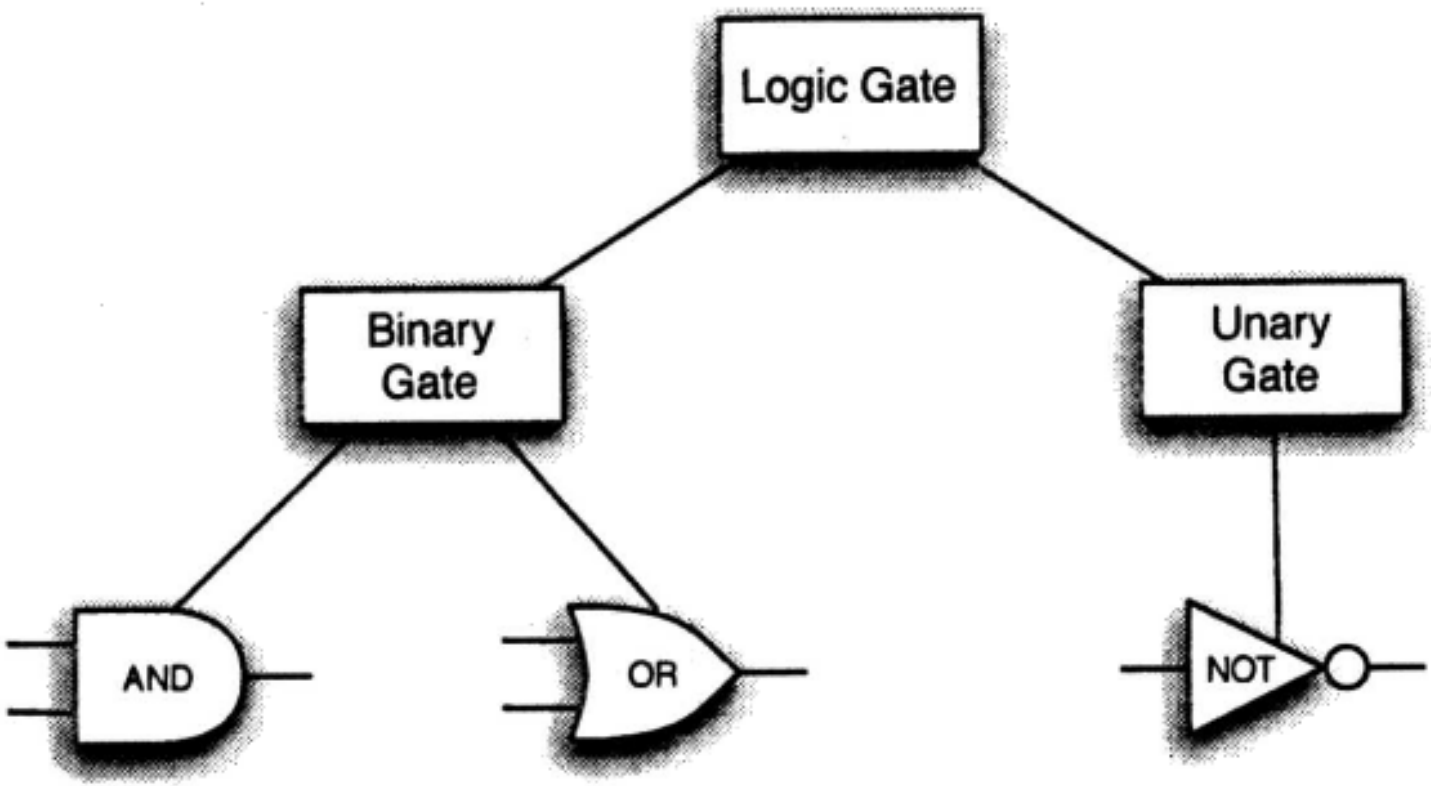


Figure 1.11: An Inheritance Hierarchy for Logic Gates

LogicGate süpersınıfı

```
class LogicGate:

    def __init__(self,n):
        self.label = n
        self.output = None

    def getLabel(self):
        return self.label

    def getOutput(self):
        self.output = self.performGateLogic()
        return self.output
```

→ her kapının performGateLogic() yöntemi farklılık gösterir

→ self parametresi: yöntemi çağıran kapı nesnesine bir referanstır

LogicGate:BinaryGate sınıfı

→ NOT kapısı UnaryGate, AND kapısı BinaryGate

→ her ikisi de LogicGate in altsınıfıdır

→ pinler: pinA, pinB

```
class BinaryGate(LogicGate):

    def __init__(self,n):
        LogicGate.__init__(self,n)

        self.pinA = None
        self.pinB = None

    def getPinA(self):
        return input("Enter Pin A input for gate "+ \
                                self.getLabel()+"-->")

    def getPinB(self):
        return input("Enter Pin B input for gate "+ \
                                self.getLabel()+"-->")
```

LogicGate:UnaryGate sınıfı

→ pin: pin

```
class UnaryGate(LogicGate):

    def __init__(self,n):
        LogicGate.__init__(self,n)

        self.pin = None

    def getPin(self):
        return input("Enter Pin input for gate "+ \
                                self.getLabel()+"-->")
```

LogicGate:BinaryGate:AndGate sınıfı

→ performGateLogic yöntemine dikkat

→ Tasarım

```
class AndGate(BinaryGate):

    def __init__(self,n):
        BinaryGate.__init__(self,n)

    def performGateLogic(self):

        a = self.getPinA()
        b = self.getPinB()
        if a==1 and b==1:
            return 1
        else:
            return 0
```

devam

→ Test

```
>>> g1 = AndGate("G")
>>> g1.getOutput()
Enter Pin A input for gate G-->1
Enter Pin B input for gate G-->0
0
>>>
```

Connector sınıfı

→ kapı hiyerarşisinde yer almaz

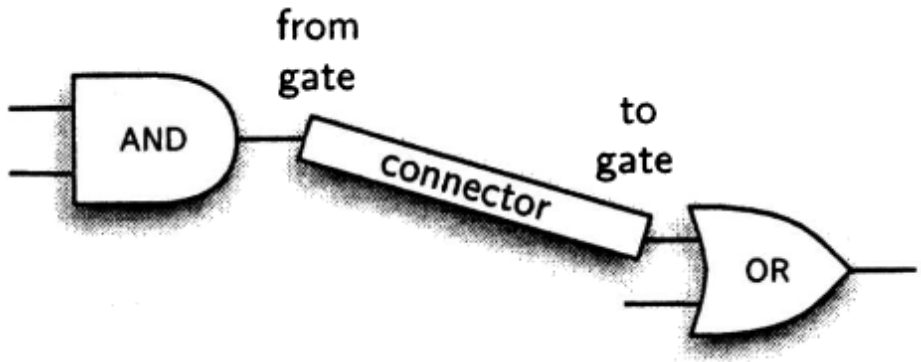


Figure 1.12: A Connector Connects the Output of One Gate to the Input of Another

HAS-A ilişkisi

HAS-A: *has-a is a relationship where one object (often called the composited object) "belongs" to (is a part or member of) another object*

→ "Connector HAS-A LogicGate": connectors LogicGate in örneklerine sahip,

→ fakat hiyerarşide yer almaz.

→ Miras **yok**.

Connector sınıfı

→ fromGate, toGate, setNextPin

```
class Connector:

    def __init__(self, fgate, tgate):
        self.fromgate = fgate
        self.togate = tgate

        tgate.setNextPin(self)

    def getFrom(self):
        return self.fromgate

    def getTo(self):
        return self.togate
```

setNextPin yöntemi

```
def setNextPin(self,source):
    if self.pinA == None:
        self.pinA = source
    else:
        if self.pinB == None:
            self.pinB = source
        else:
            print "Cannot Connect: NO EMPTY PINS"
```

Modifiye edilmiş getPin yöntemi

```
def getPinA(self):
    if self.pinA == None:
        return input("Enter Pin A input for gate "+ \
                                self.getName()+"-->")
    else:
        return self.pinA.getFrom().getOutput()
```

Circuit sınıfının tamamı

$\longrightarrow$ VIM de kod gösterimi

Örnek: devre

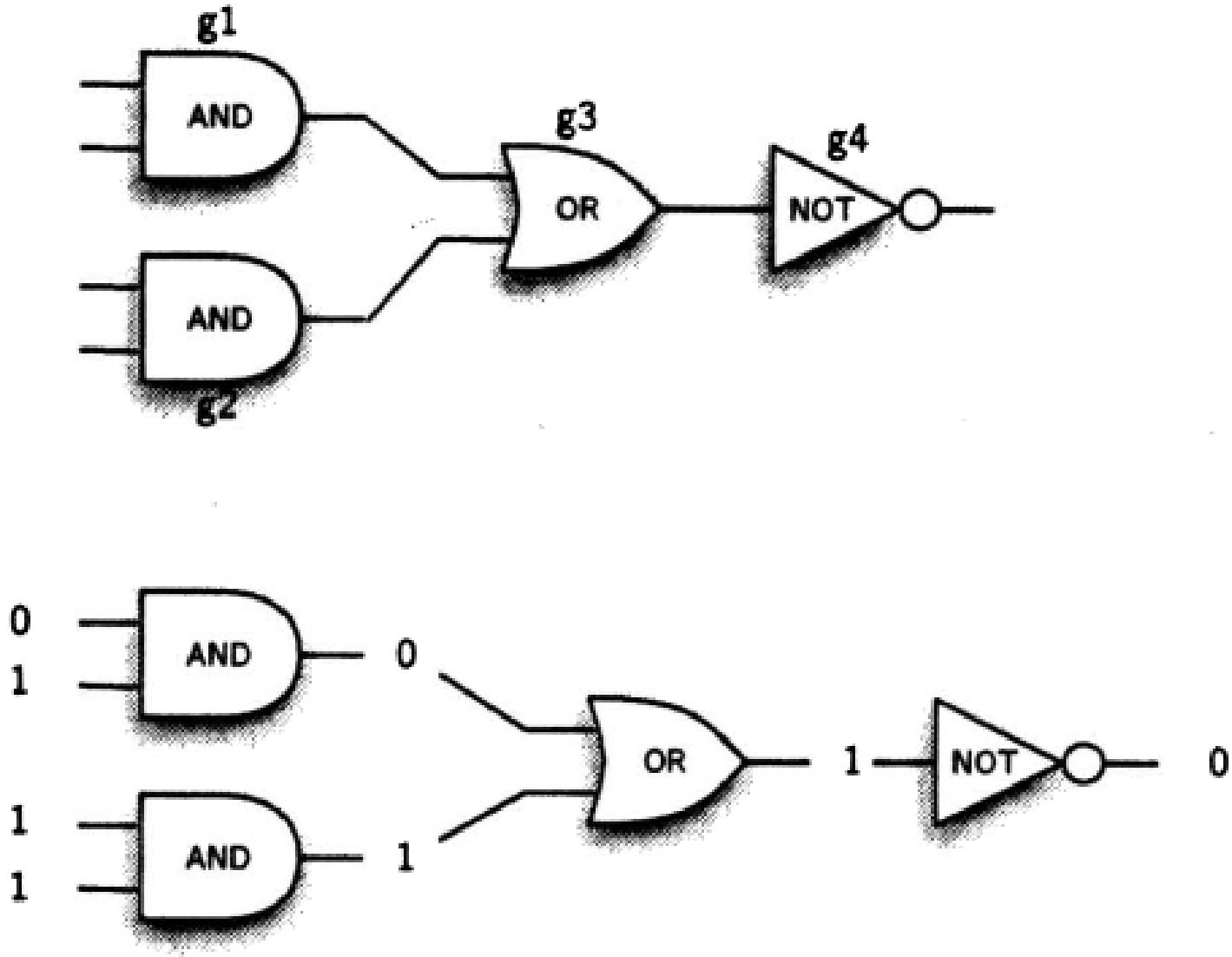


Figure 1.10: Circuit

devam (kod)

```
>>> g1 = AndGate("G1")
>>> g2 = AndGate("G2")
>>> g3 = OrGate("G3")
>>> g4 = NotGate("G4")
>>> c1 = Connector(g1, g3)
>>> c2 = Connector(g2, g3)
>>> c3 = Connector(g3, g4)
>>>
>>> g4.getOutput()
Enter Pin A input for gate G1-->0
Enter Pin B input for gate G1-->1
Enter Pin A input for gate G2-->1
Enter Pin B input for gate G2-->1
0
>>>
```

1.5 Özet

- → Bilg.Bil., problem-çözme çalışmasıdır
- → Bilg.Bil., hem veriyi hem de veriyi temsil etmek için soyutlama aracını kullanır
- → Soyut Veri Türleri, programcının problem düzleminin karmaşıklığını, veriye dair ayrıntıları gizler
- → Python oldukça güçlü, kullanımı kolay, OOP dildir
- → Python, çalıştırılabilir sözde kod
- → Listeler, tuples ve katarlar Python'un yerleşik ardışıl koleksiyonlarıdır
- → Sözlükler, ardışıl olmayan veri koleksiyonudur
- → Sınıflar, soyut veri türü gerçekleme
- → Override X yenisini yazmak
- → Sınıf: hiyerarşi - miras alma
- → Sınıf:kurucu, devam etmeden önce ebeveynin kurucusunu çağırır

Anahtar Kelimeler

→ Soyut Veri Türü, Sınıf, Veri Yapısı, Sarmalama (Encapsulation), Miras, IS-A ilişkisi Nesne, Self, Doğruluk Tablosu, Soyutlama, Hesaplanabilirlik, Deep Equality, HAS-A ilişkisi, Miras Hiyerarşisi, Yöntem, Yöntemsel Soyutlama, Shallow Equality, Algoritma, Veri Türü, Sözlük, Bilgi Gizleme, Arayüz, Değiştirebilirlik (Mutuability), Programlama, Simülasyon

1.7 Tartışma Soruları

- → Üniversite yerleşkesindeki kişiler için bir sınıf hiyerarşisi kurun.
  - → *Öğretim üyesi, memur, personel, öğrenciler yer alsın.*
  - → *Ortak olan nedir?*
  - → *Ayırt edici özellikler nelerdir?*
- → Banka hesabı için bir sınıf hiyerarşisi kurun
- → Farklı türdeki bilgisayarlar için bir sınıf hiyerarşisi kurun
- → Bu bölümde verilen sınıfları kullanarak, etkileşimli olarak devre kurup, onu test edin.

devam