Yöntem ile Programlamaya Giriş

ÖNDER ÖRNEK

Yöntem ile Programlamaya Giriş

BİRİNCİ BASKI
2026

Yöntem ile Programlamaya Giriş
Önder Örnek

Telif hakkı © 2026 Önder Örnek.

Bu eser Creative Commons Atıf 4.0 Uluslararası Lisansı (CC BY 4.0) ile lisanslanmıştır. Yazarın adı belirtilerek, lisansa bağlantı verilerek ve yapılan değişiklikler gösterilerek ticari amaçlar dahil her ortamda ve biçimde kopyalanabilir, dağıtılabilir ve uyarlanabilir. Lisans metni:
`https://creativecommons.org/licenses/by/4.0/deed.tr`

Eserdeki örnek programlar MIT lisansıyla kullanılabilir. Yöntem yazılımı da MIT lisanslıdır.

Birinci baskı: Kasım 2026

Dizgi: Metin TeX Gyre Pagella, program kodları DejaVu Sans Mono yazı tipiyle, XeLaTeX kullanılarak dizilmiştir.

İçindekiler

Önsöz vii

Başlamadan önce 1
Kurulum 1
İlk program 1
Etkileşimli kabuk 2

1 Program yapısı 3

2 Değişkenler, tipler ve işleçler 5
Temel tipler 6
İşleçler 7

3 Giriş ve çıkış: `yaz` ve `oku` 9
Alıştırmalar 11

4 Karar vermek: `eğer` ve `değerine göre` 13
Bir değeri seçeneklere göre ayırmak: `değerine göre` 14

5 Döngüler 17
Sayaçlı döngü: `… kadar her i için` 18
`iken` döngüsü: sayı tahmin oyunu 19
`tekrarla`, `her`, `kır` ve `devam` 20
Belirli sayıda tekrar: `kez tekrarla` 21
Alıştırmalar 21

6 Prosedürler ve fonksiyonlar 23

7 Metinler, listeler, sözlükler ve diziler 27
Metinler 27
Listeler 28
Sözlükler 29
Boyutu sabit listeler 30

8 Fonksiyonlar değer olarak: lambda ve boru işleci 33
Alıştırmalar 35

9 Kayıtlar ve sınıflar 37
Kayıtlar 37
Sınıflar 38
Kalıtım 40

10 Hatalarla başa çıkmak 43

11 Birimler: programı parçalara bölmek 45
Alıştırmalar 47

12 Tensörler ve otomatik türev 49
Tensörlerle işlem 49
Otomatik türev ve gradyan inişi 50

13 Sinir ağları 53
İlk sinir ağı: XOR 53
Kendi eğitim döngünüz: üç sınıflı sınıflandırma 55

14 Klasik yapay zekâ: arama, uzman sistemler, dil işleme 57
Arama algoritmaları 57
Uzman sistem 58
Metin sınıflandırma: spam filtresi 59
Alıştırmalar 60

15 Kuantum programlama 61
İlk kübit 61
Dolanıklık 62
Temel kapılar 63
Kuantum araması: Grover algoritması 64
Kuantum ve yapay zekâ buluşuyor: değişimsel devreler 64
Alıştırmalar 65

A Anahtar sözcükler 67

B Sık yapılan hatalar 69

C Alıştırma çözümleri 71

Önsöz

Bu kitap, programlamayı hiç bilmeyenlere ve Yöntem dilini öğrenmek isteyen programcılara adım adım yol gösterir. On beş bölümün sonunda kendi programlarınızı, sınıflarınızı, birimlerinizi yazabilecek; küçük bir sinir ağı eğitebilecek ve bir kuantum devresi kurabileceksiniz.

Yöntem nedir? Türkçe anahtar sözcükleri olan, Turbo Pascal soyundan gelen, nesne tabanlı ve modüler bir programlama dilidir. Pascal'ın açık ve okunaklı yapısını korur; yanına yapay zekâ için otomatik türevli tensörler ve ayrı bir kuantum birimi ekler.

Kitabı nasıl okuyabilirsiniz?

- Bölümleri sırayla okumanızı öneririz; her bölüm bir öncekinin üzerine kurulur.
- Her örneği kendiniz yazıp çalıştırmanız, konuyu pekiştirmenin en iyi yoludur. Kitaptaki tüm programlar yorumlayıcıda denenmiştir; altlarında gerçek çıktıları yer alır.
- Bölüm sonundaki alıştırmaları yapmanız yararlı olacaktır. Çözümleri kitabın sonundaki ekte bulabilirsiniz.
- Takıldığınızda Ek B'deki **Sık yapılan hatalar** bölümüne bakabilirsiniz.

Başlamadan önce

Kurulum

Hiçbir şey kurmadan başlamak isterseniz programları tarayıcıdaki **oyun alanında** yazıp çalıştırabilirsiniz. Kendi bilgisayarınızda çalışmak için önerilen yol, **JavaScript derleyicisini** kurmaktır; bunun için **Node.js 18 veya üstü** gerekir.

1. Önce derleyiciyi kurabilirsiniz: `npm install -g yontem`
2. Ardından kurulumu sınayabilirsiniz: `yontem --surum`

Adımların ayrıntılarını ve işletim sisteminize özgü notları *Yöntem Kurulum Kılavuzu*'nda bulabilirsiniz. Kitaptaki örnek programlar Yöntem'in deposundadır: `git clone https://github.com/onderornek/yontem.git`

Node.js kuramıyorsanız Yöntem'in Python yorumlayıcısını kullanabilirsiniz. Yorumlayıcı, Yöntem'in deposuyla gelir, yalnızca **Python 3.8 veya üstünü** ister ve kurulum gerektirmez: bu kitapta `yontem program.y` yazan her yerde, depo klasöründe `python3 yontem.py program.y` yazmanız yeterlidir. İki uygulama aynı çıktıyı verir; yorumlayıcı yalnızca daha yavaştır.

Yöntem programları `.y` uzantılı düz metin dosyalarıdır. Herhangi bir metin düzenleyicisiyle (VS Code, Not Defteri, TextEdit) yazabilirsiniz. Türkçe karakterlerin bozulmaması için dosyayı **UTF-8** olarak kaydetmeniz gerekir.

İlk program

`ilk.y` adında bir dosya oluşturup şunu yazalım:

```
program İlkProgram
{ Benim ilk Yöntem programım }
başla
  yaz('Merhaba, Dünya!', 1)
  yaz('Yöntem ile programlamaya hoş geldiniz.', 1)
bitir.
```

Şimdi çalıştıralım:

```
$ yontem ilk.y
Merhaba, Dünya!
Yöntem ile programlamaya hoş geldiniz.
```

Satır satır inceleyelim:

- `program İlkProgram` programa bir ad verir. Bu satır isteğe bağlıdır ama iyi bir alışkanlıktır.
- `{ ... }` arası **yorumdur**. Bilgisayar yorumları okumaz; onlar insanlar içindir. `// satır sonuna kadar` ve `(* ... *)` da yorumdur.
- `başla` ile `bitir.` programın gövdesini çevreler. Programın sonundaki **noktayı** unutmamak gerekir.
- `yaz('...', 1)` metni ekrana yazar. İkinci parametre olan `1`, yazdıktan sonra kaç kez alt satıra geçileceğini söyler.
- Her komut kendi satırına yazılır. Noktalı virgül (`;`) yalnızca aynı satıra birden çok komut yazıldığında gerekir.

Etkileşimli kabuk

Kısa denemeler için dosya yazmanız gerekmez. Etkileşimli kabuk Python yorumlayıcısıyla gelir: depo klasöründe `python3 yontem.py` komutunu tek başına çalıştırınca kabuk açılır. Bir ifade yazdığınızda değeri hemen gösterilir:

```
y> 6 * 7
42
y> 'Yöntem'.BüyükHarf
'YÖNTEM'
y> çıkış
```

BÖLÜM

1

Program yapısı

Her Yöntem programı aynı iskeleti izler: önce ne kullanacağınızı **bildirirsiniz**, sonra `başla … bitir.` arasında ne yapılacağını yazarsınız.

```
program Ad              // 1. program adı (isteğe bağlı)
kullan Matematik        // 2. kullanılacak birimler
sabit    ...             // 3. bildirimler: sabitler,
tip      ...             //    tipler,
değişken ...             //    değişkenler,
fonksiyon ...            //    alt programlar
başla                    // 4. ana gövde
  ...
bitir.
```

Bu düzen Pascal'dan gelir ve büyük programlarda bile neyin nerede olduğunu bulmayı kolaylaştırır.

Büyük/küçük harf fark etmez. `Başla`, `BAŞLA` ve `basla` aynı sözcüktür. Ayrıca Türkçe harfler ASCII karşılıklarıyla eş sayılır: Türkçe klavyeniz yoksa `degisken`,

`eger`, `icin` yazabilirsiniz. Bunun bir sonucu vardır: `sayac` ile `sayaç`, `x` ile `X` **aynı addır**. Aynı yerde ikisini birden tanımlarsanız Yöntem sizi uyarır.

BÖLÜM

2

Değişkenler, tipler ve işleçler

Değişken, bir değeri saklayan adlandırılmış bir kutudur. Yöntem'de değişkenler kullanılmadan önce `değişken` bölümünde tipleriyle bildirilir, sonra `:=` ile değer atanır.

```
program Değişkenler

sabit
  KDV_ORANI: ondalık = 0.20          // sabitler değiştirilemez

değişken
  ad: metin
  yaş: tamsayı
  boy: ondalık
  öğrenci: mantıksal
  fiyat: ondalık = 250.0     // ilk değerli bildirim

başla
```

```
  ad := 'Elif'
  yaş := 21
  boy := 1.68
  öğrenci := doğru

  yaz('Ad      : ' + ad, 1)
  yaz('Yaş     : ' + yaş, 1)
  yaz('Boy     : ' + boy + ' m', 1)
  yaz('Öğrenci : ' + öğrenci, 1)
  yaz('KDV''li fiyat: ' + fiyat * (1 + KDV_ORANI), 1)
  yaz('Tipler: ' + Tür(yaş) + ', ' + Tür(boy) + ', ' + Tür(ad), 1)
bitir.
```

```
Ad      : Elif
Yaş     : 21
Boy     : 1.68 m
Öğrenci : doğru
KDV'li fiyat: 300.0
Tipler: tamsayı, ondalık, metin
```

Dikkat edilecek noktalar şunlardır:

- Atama `:=` ile yapılır. Tek başına `=` karşılaştırmadır.
- Metinler tek tırnak arasında yazılır. Metnin içinde tırnak gerekiyorsa iki kez yazılır: `'KDV''li'`.
- `+` işlecinin bir yanı metinse, diğer yanı da metne çevrilir. Bu yüzden `'Yaş: ' + yaş` çalışır.

Temel tipler

Tip	Örnek değer	Varsayılan	Açıklama
`tamsayı`	`42`, `-7`, `$FF`	`0`	Tam sayılar, sınırsız büyüklükte
`ondalık`	`3.14`, `1e-3`	`0.0`	Kesirli sayılar
`metin`	`'Merhaba'`	`''`	Yazılar; karakterler 1'den numaralanır
`mantıksal`	`doğru`, `yanlış`	`yanlış`	Evet/hayır değerleri
`liste`	`[1, 2, 3]`	`[]`	Bölüm 7'de anlatılır
`sözlük`	`['a': 1]`	`[:]`	Bölüm 7'de anlatılır
`herhangi`	her şey	`boş`	Tip denetimi yapılmaz

Yöntem tipleri **çalışma sırasında denetler**. Yanlış tipte bir değer atarsanız program açık bir mesajla durur:

```
değişken n: tamsayı
başla
  n := 3.5
bitir.
```

```
Çalışma hatası [b3h.y, satır 3] 'n' için tip uyuşmazlığı: tamsayı bekleniyordu, ondalık
    ↪ verildi
```

Tipler arasında dönüşüm yapmak için tip adlarını fonksiyon gibi kullanabilirsiniz:

```
program Dönüşümler
değişken
  s: metin
  n: tamsayı
  x: ondalık
başla
  s := '42'
  n := tamsayı(s) + 8
  x := ondalık('3,75')          // virgül de kabul edilir
  yaz(n, 1)
  yaz(x * 2, 1)
  yaz(metin(n) + metin(n), 1)
  yaz(Yuvarla(3.6) + ' ' + Kes(3.6), 1)
bitir.
```

```
50
7.5
5050
4 3
```

İşleçler

```
program İşleçler
başla
  yaz('17 + 5   = ' + (17 + 5), 1)
  yaz('17 / 5   = ' + (17 / 5), 1)
  yaz('17 böl 5 = ' + (17 böl 5), 1)
  yaz('17 mod 5 = ' + (17 mod 5), 1)
  yaz('2 ** 10  = ' + (2 ** 10), 1)
```

```
  yaz('(3 > 2) ve (1 = 1) = ' + ((3 > 2) ve (1 = 1)), 1)
  yaz('değil doğru = ' + (değil doğru), 1)
  yaz('5 > 3 ve 2 > 8 = ' + (5 > 3 ve 2 > 8), 1)
bitir.
```

```
17 + 5   = 22
17 / 5   = 3.4
17 böl 5 = 3
17 mod 5 = 2
2 ** 10  = 1024
(3 > 2) ve (1 = 1) = doğru
değil doğru = yanlış
5 > 3 ve 2 > 8 = yanlış
```

İşleç	Anlamı
`+ - *`	toplama, çıkarma, çarpma
`/`	bölme; sonuç her zaman ondalık
`böl`, `mod`	tam sayı bölmesi ve kalan
**	üs alma
`= <> < > < = > =`	eşit, eşit değil, küçük, büyük...
`ve`, `veya`, `yada`, `değil`	mantıksal işleçler
`+ = - = * = /=`	kısa atama: `x + = 1` ≡ `x := x + 1`

İpucu: Bir ifadeyi metnin içine koyarken paranteze almanız iyi olur. `'Toplam: ' + 2 + 3` önce `'Toplam: 2'` olur, sonuç da `'Toplam: 23'` çıkar. Doğrusu `'Toplam: ' + (2 + 3)` biçimidir.

BÖLÜM

3

Giriş ve çıkış: `yaz` ve `oku`

Yöntem'de ayrı bir "satır sonu" komutu yoktur. `yaz` ve `oku` komutlarının ikinci parametresi kaç satır ilerleneceğini söyler.

Komut	Ne yapar?
`yaz(değer)`	Değeri yazar, aynı satırda kalır
`yaz(değer, 1)`	Yazar ve alt satıra geçer
`yaz('test', 3)`	Yazar ve 3 kez alt satıra geçer
`yaz('', 1)`	Boş bir satır yazar
`oku(x)`	`x` sayıysa satırdaki sıradaki sayıyı, metinse satırın kalanını okur
`oku(x, 1)`	Okur ve sonraki satıra geçer

```
program Tanışma
değişken
  ad: metin
  doğumYılı, yaş: tamsayı
başla
  yaz('Adınız: ')
  oku(ad, 1)
  yaz('Doğum yılınız: ')
  oku(doğumYılı, 1)
  yaş := 2026 - doğumYılı
  yaz('Merhaba ' + ad + ', bu yıl ' + yaş + ' yaşındasınız.', 2)
  yaz('Güle güle!', 1)
bitir.
```

```
Adınız: Elif
Doğum yılınız: 2005
Merhaba Elif, bu yıl 21 yaşındasınız.

Güle güle!
```

`yaz(..., 2)` iki kez alt satıra geçtiği için araya boş bir satır girdi. `oku`, bilgiyi değişkenin tipine göre çevirir: `doğumYılı` tamsayı olduğu için yazılan `2005` sayıya dönüştü.

Aynı satırdan birden çok sayı okumak için satır parametresini yalnızca sonuncusuna vermeniz yeterlidir:

```
program İkiSayı
değişken
  a, b: tamsayı
başla
  yaz('Aynı satıra iki sayı yazın: ')
  oku(a)
  oku(b, 1)
  yaz(a + ' + ' + b + ' = ' + (a + b), 1)
  yaz(Biçimle('Ortalama: {:.2}', (a + b) / 2), 1)
  yaz((a / b):10:3, 1)
bitir.
```

```
Aynı satıra iki sayı yazın: 7 3
7 + 3 = 10
Ortalama: 5.00
     2.333
```

Sayıları biçimlendirmenin iki yolu vardır:

- **Turbo Pascal biçimi:** `yaz(x:10:3)` sayıyı 10 karakter genişliğe, 3 ondalık basamakla yazar.
- **`Biçimle`:** `{}` yerlerine değerleri koyar. `{:.2}` iki basamak, `{:8}` 8 karakter genişlik, `{:8.2}` ikisi birden.

Alıştırmalar

1. Kullanıcıdan bir dairenin yarıçapını okuyup alanını ve çevresini iki basamakla yazan bir program yazınız. (`pi` yerleşik bir sabittir.)
2. Saniye cinsinden verilen bir süreyi saat, dakika ve saniyeye çeviren bir program yazınız. `böl` ve `mod` işleçlerinden yararlanabilirsiniz. Örnek: 3725 → 1 saat 2 dakika 5 saniye.

BÖLÜM

4

Karar vermek: `eğer` *ve* `değerine göre`

Programlar koşullara göre farklı yollar izler. `eğer koşul ise A değilse B` yapısı koşul doğruysa A'yı, değilse B'yi çalıştırır. Koşul mutlaka `mantıksal` bir değer olmalıdır.

```
program NotHesabı
değişken
  puan: tamsayı
  harf: metin
başla
  yaz('Sınav puanı (0-100): ')
  oku(puan, 1)

  eğer (puan < 0) veya (puan > 100) ise
    yaz('Geçersiz puan!', 1)
  değilse eğer puan > = 85 ise
    harf := 'AA'
  değilse eğer puan > = 70 ise
```

```
    harf := 'BB'
  değilse eğer puan > = 50 ise
    harf := 'CC'
  değilse
    harf := 'FF'

  eğer harf <> '' ise
  başla
    yaz('Harf notu: ' + harf, 1)
    yaz(eğer harf = 'FF' ise 'Kaldınız.' değilse 'Geçtiniz!', 1)
  bitir
bitir.
```

```
Sınav puanı (0-100): 77
Harf notu: BB
Geçtiniz!
```

Bu programdan çıkarılacak dersler şunlardır:

- Bir dalda birden çok komut çalışsın istiyorsanız onları `başla … bitir` arasına almanız gerekir.
- `değilse eğer` zinciriyle çok seçenekli kararlar yazılır. İlk doğru koşul kazanır.
- `değilse` yeni bir satırda başlayabilir; komutların sonuna noktalı virgül koymak gerekmez.
- `eğer … ise … değilse …` bir **ifade** olarak da kullanılabilir ve son satırdaki gibi bir değer üretir. Bu kullanımda `değilse` zorunludur.

Bir değeri seçeneklere göre ayırmak: `değerine göre`

Tek bir değerin pek çok olası durumu varsa `x değerine göre … bitir` yapısı daha okunaklıdır. Etiketler tek değer (`1:`), virgülle liste (`6, 7:`) ya da aralık (`1..5:`) olabilir. Hiçbir etiket tutmazsa `değilse:` kolu çalışır.

```
program Günler
değişken
  gün: tamsayı
başla
  1'den 7'ye kadar her gün için
  başla
    yaz(gün + ': ')
    gün değerine göre
      1: yaz('Pazartesi')
      2: yaz('Salı')
      3: yaz('Çarşamba')
```

```
      4: yaz('Perşembe')
      5: yaz('Cuma')
      değilse: yaz('Hafta sonu')
    bitir
    gün değerine göre
      1..5: yaz(' (iş günü)', 1)
      6, 7: yaz(' (tatil)', 1)
    bitir
  bitir
bitir.
```

```
1: Pazartesi (iş günü)
2: Salı (iş günü)
3: Çarşamba (iş günü)
4: Perşembe (iş günü)
5: Cuma (iş günü)
6: Hafta sonu (tatil)
7: Hafta sonu (tatil)
```

BÖLÜM

5

Döngüler

Döngüler bir işi tekrar tekrar yapar. Yöntem'de beş döngü vardır ve hepsi Türkçe cümle düzenine uyar: önce neyin üzerinde dönüleceği, en sonda eylem gelir.

Döngü	Ne zaman kullanılır?
`1'den 10'a kadar her i için …`	Bir sayaç belli bir aralıkta dolaşacaksa
`liste içindeki her x için …`	Bir listenin, metnin ya da sözlüğün her elemanı için
`5 kez tekrarla …`	Bir iş belli sayıda yapılacaksa
`koşul iken …`	Koşul doğru olduğu sürece; hiç çalışmayabilir
`tekrarla … koşul olana dek`	Koşul doğru olana dek; en az bir kez çalışır

Ekler. Sayaçlı döngüde başlangıç değeri ayrılma ekiyle (`1'den`, `3'ten`, `n'den`), bitiş değeri yönelme ekiyle (`10'a`, `7'ye`, `n'e`) yazılır. Ek, kesme işaretiyle sayıya,

ada ya da kapanış parantezine bitişik yazılır: `(n - 1)'e`. Yöntem ses uyumunu denetlemez, ama doğru eki yazmak programı okunur kılar.

Gövde kuralı. Döngü başlığından sonra gelen tek komut döngünün gövdesidir. Birden çok komut tekrarlanacaksa `başla … bitir` arasına alınır. `tekrarla … olana dek` kendi sınırlarını taşıdığı için `başla` istemez.

Sayaçlı döngü: `… kadar her i için`

```
program ÇarpımTablosu
değişken
  i, j: tamsayı
başla
  1'den 5'e kadar her i için
  başla
    1'den 5'e kadar her j için
      yaz(Biçimle('{:4}', i * j))
    yaz('', 1)
  bitir
  yaz('Geri sayım: ')
  10'dan 0'a adımda 2 azalan her i için
    yaz(i + ' ')
  yaz('Ateş!', 1)
bitir.
```

```
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
   5  10  15  20  25
Geri sayım: 10 8 6 4 2 0 Ateş!
```

İç içe iki döngü bir tablo üretti. `{:4}` her sayıyı 4 karakterlik alana sağa yaslayarak yazdı. Son döngüdeki `adımda 2 azalan`, sayacın ikişer ikişer azalacağını söyler.

Sayan döngünün dört biçimi vardır:

Yazım	Ne yapar
`1'den 10'a kadar her i için`	Birer birer artar
`10'dan 1'e kadar her i için`	İki sınır da sabit sayıysa yön sınırlardan anlaşılır: birer birer azalır
`a'dan b'ye adımda c artan her i için`	c'şer artar
`b'den a'ya adımda c azalan her i için`	c'şer azalır

Sınırlardan biri değişkense `kadar` her zaman artarak sayar; başlangıç bitişten büyükse döngü hiç çalışmaz. Değişken sınırlarla geriye saymak için yönü açıkça yazmanız gerekir: `n'den 1'e adımda 1 azalan her i için`. Adım her zaman pozitif bir tamsayıdır; yönü `artan` ya da `azalan` sözcüğü belirler.

`iken` döngüsü: sayı tahmin oyunu

```
program SayıTahmini
değişken
  gizli, tahmin, deneme: tamsayı
başla
  Rastgeleştir(7)              // her çalıştırmada aynı sayı (deneme için)
  gizli := Rastgele(1, 100)
  deneme := 0
  tahmin := -1
  tahmin <> gizli iken
  başla
    yaz('Tahmininiz: ')
    oku(tahmin, 1)
    deneme + = 1
    eğer tahmin < gizli ise yaz('Daha büyük.', 1)
    değilse eğer tahmin > gizli ise yaz('Daha küçük.', 1)
  bitir
  yaz('Bildiniz! ' + deneme + ' denemede.', 1)
bitir.
```

```
Tahmininiz: 50
Daha küçük.
Tahmininiz: 25
Daha büyük.
Tahmininiz: 37
Daha büyük.
Tahmininiz: 43
Daha küçük.
Tahmininiz: 40
Daha büyük.
Tahmininiz: 42
Bildiniz! 6 denemede.
```

`Rastgele(1, 100)` 1 ile 100 arasında bir tam sayı üretir. `Rastgeleştir(7)` üreteci sabit bir tohumla başlatır; böylece program her seferinde aynı sayıyı seçer. Gerçek oyunda bu satırı `Rastgeleştir;` yapabilirsiniz.

`tekrarla`, `her`, `kır` ve `devam`

```
program Döngüler2
değişken
  n, toplam, i: tamsayı
  meyve: metin
başla
  // tekrarla: gövde en az bir kez çalışır
  n := 9875
  toplam := 0
  tekrarla
    toplam := toplam + n mod 10
    n := n böl 10
  n = 0 olana dek
  yaz('9875 sayısının basamakları toplamı: ' + toplam, 1)

  // her: bir koleksiyonun elemanları üzerinde dolaşır
  ['elma', 'armut', 'kiraz'] içindeki her meyve için
    yaz(meyve.BüyükHarf + ' ')
  yaz('', 1)

  // kır ve devam
  1'den 20'ye kadar her i için
  başla
    eğer i mod 3 = 0 ise devam    // 3'ün katlarını atla
    eğer i > 10 ise kır           // 10'u geçince döngüden çık
    yaz(i + ' ')
  bitir
  yaz('', 1)
bitir.
```

```
9875 sayısının basamakları toplamı: 29
ELMA ARMUT KİRAZ
1 2 4 5 7 8 10
```

- `tekrarla … olana dek` gövdesinde birden çok komut olabilir, `başla … bitir` gerekmez.
- `devam` döngünün o turunu atlayıp bir sonrakine geçer; `kır` döngüyü tamamen bitirir.
- `BüyükHarf` Türkçe kurallarını bilir: `kiraz` → `KİRAZ`.

Belirli sayıda tekrar: `kez tekrarla`

```
program Yıldızlar
değişken
  satır: tamsayı
başla
  1'den 4'e kadar her satır için
  başla
    satır kez tekrarla yaz('*')
    yaz('', 1)
  bitir
bitir.
```

```
*
**
***
****
```

`satır kez tekrarla yaz('*')` yıldızı satırın numarası kadar yazar. Bir sayaca ihtiyacınız yoksa en kısa ve en doğal biçim budur.

Alıştırmalar

1. 1'den 100'e kadar sayıları yazan, ama 3'ün katları yerine `Fizz`, 5'in katları yerine `Buzz`, ikisinin de katları yerine `FizzBuzz` yazan bir program yazınız.
2. Kullanıcıdan 0 girilene kadar sayılar okuyup en sonda toplamı ve ortalamayı yazan bir program yazınız.

BÖLÜM

6

Prosedürler ve fonksiyonlar

Büyük bir işi küçük, adlandırılmış parçalara bölmek programı okunur ve yeniden kullanılabilir yapar. **Prosedür** bir iş yapar; **fonksiyon** ise bir değer döndürür.

```
program AltProgramlar

prosedür Çizgi(uzunluk: tamsayı = 20; karakter: metin = '-')
değişken
  i: tamsayı
başla
  1'den uzunluk'a kadar her i için yaz(karakter)
  yaz('', 1)
bitir

fonksiyon Kare(x: ondalık): ondalık
başla
  sonuç := x * x
bitir
```

```
fonksiyon Faktöriyel(n: tamsayı): tamsayı
başla
  eğer n < = 1 ise dön 1                  // dön: değeri ver ve hemen çık
  dön n * Faktöriyel(n - 1)              // özyineleme
bitir

prosedür Takas(değişken a, b: tamsayı)   // referansla parametre
değişken
  geçici: tamsayı
başla
  geçici := a
  a := b
  b := geçici
bitir

değişken
  x, y: tamsayı

başla
  Çizgi
  yaz('Kare(1.5)    = ' + Kare(1.5), 1)
  yaz('Faktöriyel(20) = ' + Faktöriyel(20), 1)
  x := 1;  y := 2
  Takas(x, y)
  yaz('Takas sonrası: x = ' + x + ', y = ' + y, 1)
  Çizgi(10, '=')
bitir.
```

```
--------------------
Kare(1.5)    = 2.25
Faktöriyel(20) = 2432902008176640000
Takas sonrası: x = 2, y = 1
==========
```

Kavram	Örnekte	Açıklama
Varsayılan parametre	`uzunluk: tamsayı = 20`	Verilmezse bu değer kullanılır. `Çizgi;` boş çağrılabildi.
Yerel değişken	`değişken i`	Yalnızca o alt programın içinde yaşar
`sonuç`	`sonuç := x * x`	Fonksiyonun döndüreceği değer
`dön`	`dön 1`	Değeri belirler ve fonksiyondan hemen çıkar

Kavram	Örnekte	Açıklama
Özyineleme	`Faktöriyel(n - 1)`	Fonksiyon kendini çağırır; bir durma koşulu şarttır
Referans parametre	`değişken a, b`	Değişikliği çağıranın değişkenine yansıtır

Normal parametreler **değer olarak** aktarılır: fonksiyon kendi kopyasını değiştirir, asıl değişken etkilenmez. Başına `değişken` yazılan parametre ise çağıranın değişkenine bağlanır; `Takas` bu yüzden çalıştı. Parametresiz alt programlar parantezsiz çağrılır: `Çizgi;`

BÖLÜM

7

Metinler, listeler, sözlükler ve diziler

Tek bir değer yerine bir değer **topluluğu** ile çalışmak için koleksiyonlar kullanılır. Bir kuralı en baştan akılda tutmakta yarar vardır: **metinler 1'den, listeler 0'dan numaralanır.** İlki Turbo Pascal geleneğidir, ikincisi ise modern dillerin alışkanlığıdır.

Metinler

```
program Metinler
değişken
  s: metin
  kelimeler: liste
  i: tamsayı
başla
  s := 'Yöntem dili'
  yaz('Uzunluk      : ' + Uzunluk(s), 1)
  yaz('İlk harf     : ' + s[1], 1)              // metinler 1'den başlar
```

```
  yaz('Son harf     : ' + s[Uzunluk(s)], 1)
  yaz('AltMetin     : ' + AltMetin(s, 8, 4), 1)
  yaz('Konum(''dil'') : ' + Konum('dil', s), 1)
  yaz('Büyük harf   : ' + s.BüyükHarf, 1)
  yaz('Değiştir     : ' + Değiştir(s, 'dili', 'programı'), 1)
  kelimeler := Ayır('elma,armut,kiraz', ',')
  yaz('Ayır         : ' + kelimeler, 1)
  yaz('Birleştir    : ' + Birleştir(kelimeler, ' + '), 1)
  yaz('Tersten      : ')
  Uzunluk(s)'den 1'e adımda 1 azalan her i için yaz(s[i])
  yaz('', 1)
bitir.
```

```
Uzunluk      : 11
İlk harf     : Y
Son harf     : i
AltMetin     : dili
Konum('dil') : 8
Büyük harf   : YÖNTEM DİLİ
Değiştir     : Yöntem programı
Ayır         : ['elma', 'armut', 'kiraz']
Birleştir    : elma + armut + kiraz
Tersten      : ilid metnöY
```

Metin işlevlerinin çoğu iki biçimde yazılabilir: `BüyükHarf(s)` ya da `s.BüyükHarf`. Hangisi size daha okunaklı geliyorsa onu kullanabilirsiniz.

Listeler

```
program Listeler
değişken
  notlar: liste
  n: herhangi
  toplam: ondalık
başla
  notlar := [72, 85, 90, 64]
  notlar.Ekle(78)                          // sona ekle
  yaz('Notlar     : ' + notlar, 1)
  yaz('İlk not    : ' + notlar[0], 1)      // listeler 0'dan başlar
  yaz('Not sayısı : ' + notlar.Uzunluk, 1)
  yaz('Sıralı     : ' + Sırala(notlar), 1)
  yaz('En yüksek  : ' + Maks(notlar), 1)
  toplam := 0
  notlar içindeki her n için toplam + = n
  yaz('Ortalama   : ' + toplam / Uzunluk(notlar), 1)
  yaz('85 var mı? : ' + (85 içinde notlar), 1)
```

```
  yaz('1..5       : ' + [1..5], 1)
  notlar[0] := 100
  Sil(notlar, 1)
  yaz('Değişmiş   : ' + notlar, 1)
bitir.
```

```
Notlar     : [72, 85, 90, 64, 78]
İlk not    : 72
Not sayısı : 5
Sıralı     : [64, 72, 78, 85, 90]
En yüksek  : 90
Ortalama   : 77.8
85 var mı? : doğru
1..5       : [1, 2, 3, 4, 5]
Değişmiş   : [100, 90, 64, 78]
```

`Sırala` yeni, sıralı bir liste döndürür; asıl liste değişmez. `Sil(notlar, 1)` 1 numaralı elemanı (`notlar[1]`) çıkarır. `[1..5]` bir aralığı listeye çevirir.

Sözlükler

Sözlük, **anahtar → değer** çiftlerini saklar. Bir değere sıra numarasıyla değil, adıyla ulaşırsınız.

```
program Sözlükler
değişken
  başkent: sözlük
  ülke, şehir: herhangi
  sayım: sözlük
başla
  başkent := ['Türkiye': 'Ankara', 'Fransa': 'Paris', 'Japonya': 'Tokyo']
  başkent['Mısır'] := 'Kahire'              // yeni anahtar ekle
  yaz('Fransa''nın başkenti: ' + başkent['Fransa'], 1)
  yaz('Ülke sayısı: ' + Uzunluk(başkent), 1)
  başkent içindeki her ülke, şehir için
    yaz('  ' + ülke + ' -> ' + şehir, 1)
  yaz('İtalya kayıtlı mı? ' + ('İtalya' içinde başkent), 1)
  yaz('Varsayılanla al: ' + başkent.Al('İtalya', 'bilinmiyor'), 1)

  sayım := Say(Ayır('bir iki bir üç iki bir'))
  yaz('Sözcük sayıları: ' + sayım, 1)
bitir.
```

```
Fransa'nın başkenti: Paris
Ülke sayısı: 4
  Türkiye -> Ankara
  Fransa -> Paris
  Japonya -> Tokyo
  Mısır -> Kahire
İtalya kayıtlı mı? yanlış
Varsayılanla al: bilinmiyor
Sözcük sayıları: ['bir': 3, 'iki': 2, 'üç': 1]
```

- Sözlük yazımı köşeli parantez ve iki noktadır: `['anahtar': değer]`. Boş sözlük `[:]` olarak yazılır.
- Olmayan bir anahtarı okumak hatadır. Emin değilseniz `içinde` ile denetleyebilir ya da `.Al(anahtar, varsayılan)` kullanabilirsiniz.
- `sözlük içindeki her a, d için` anahtarı ve değeri birlikte verir.
- `Say` bir listedeki elemanları sayar. Metin çözümlemede çok işe yarar.

Boyutu sabit listeler

Bazı veriler baştan belli sayıda elemandan oluşur: haftanın 7 günü, yılın 12 ayı. Böyle durumlarda listenin boyutunu `Uzunluk` ile sabitleyebilir ve elemanları istediğiniz sayıdan başlayarak numaralayabilirsiniz. `sıcaklık.Uzunluk(7, 1)` listeyi 7 elemanlı yapar ve ilk elemana 1 numarasını verir; sınır dışına çıkmak anında yakalanır.

```
program SabitListe
değişken
  sıcaklık: ondalık listesi
  gün: tamsayı
başla
  sıcaklık.Uzunluk(7, 1)
  1'den 7'ye kadar her gün için
    sıcaklık[gün] := 20 + gün * 1.5
  yaz(sıcaklık, 1)
  yaz('Sınırlar: ' + sıcaklık.Alt + '..' + sıcaklık.Üst, 1)
  sıcaklık[8] := 30
bitir.
```

```
[21.5, 23.0, 24.5, 26.0, 27.5, 29.0, 30.5]
Sınırlar: 1..7
Çalışma hatası [d5.y, satır 11] Liste dizini sınır dışı: 8 (geçerli 1..7)
```

`Uzunluk` çağrısının iki biçimi vardır:

Yazım	Sonuç
`L.Uzunluk(12)`	12 elemanlı liste; numaralar 0'dan başlar (0..11)
`L.Uzunluk(12, 1)`	12 elemanlı liste; numaralar 1'den başlar (1..12)

Yeni elemanlar tipin başlangıç değeriyle dolar (sayılarda 0, metinlerde boş metin); listede önceden eleman varsa baştan itibaren korunur. Boyutu sabitlenen liste ile serbest liste arasındaki farkı akılda tutmakta yarar vardır:

	Serbest liste	Boyutu sabit liste
Bildirim	`notlar: tamsayı listesi`	`sıcaklık: ondalık listesi`, ardından `sıcaklık.Uzunluk(7, 1)`
Numaralar	0'dan başlar	Sizin verdiğiniz sayıdan başlar (burada 1..7)
Sınırlar	`0` ve `Uzunluk(notlar) - 1`	`sıcaklık.Alt` ve `sıcaklık.Üst`
Boyut	Değişir: `Ekle`, `Sil`	Sabittir; yalnızca yeni bir `Uzunluk` çağrısıyla değişir
Sona eklemek	`Ekle(notlar, 95)`	Eklenemez
Araya eklemek	`Ekle(notlar, 95, 1)` (konum 1'e)	Eklenemez
2 numaralı elemanı silmek	`notlar.Sil(2)`	Silinemez

`Ekle` iki biçimde kullanılır: `Ekle(notlar, 95)` elemanı sona ekler; `Ekle(notlar, 95, 1)` ise elemanı 1 numaralı konuma yerleştirir ve sonraki elemanları birer sıra kaydırır: önce eklenecek değer, sonra konum yazılır. Aynı işlemleri nokta ile de yazabilirsiniz: `notlar.Ekle(95)`, `notlar.Ekle(95, 1)`.

Bir elemanı silmenin üç yazımı vardır ve üçü de aynı işi yapar:

```
notlar.Sil(2)
notlar(2).Sil
notlar listesindeki 2. elemanı Sil
```

Üçü de 2 numaralı elemanı, yani `notlar[2]`'yi siler. Listede elemanlar 0'dan numaralandığı için bu, baştan üçüncü sıradaki elemandır. Konumu hesaplayarak vermek isterseniz cümle biçiminde ayraç kullanabilirsiniz: `notlar listesindeki (i + 1). elemanı Sil`. `Sil` çıkardığı elemanı döndürür; `x := notlar.Sil(0)` yazarak onu bir değişkene alabilirsiniz.

Boyutu sabitlenmiş bir listeye `Ekle` ya da `Sil` uygulamak hata verir. Boyutu değiştirmek isterseniz `Uzunluk` çağrısını yeni değerlerle yineleyebilir, serbest bir kopya gerekiyorsa `liste(sıcaklık)` yazabilirsiniz. Tip adlarında `tamsayı dizisi` ile `tamsayı listesi` aynı anlama gelir.

BÖLÜM

8

Fonksiyonlar değer olarak: lambda ve boru işleci

Yöntem'de fonksiyonlar sıradan değerlerdir: bir değişkene konabilir, başka bir fonksiyona verilebilir. Bu özellik veri işlemede, özellikle yapay zekâ programlarında çok kullanılır.

```
program Fonksiyonel
değişken
  sayılar, kareler, çiftler: liste
  genelToplam: tamsayı
  kişiler: liste
  kare: herhangi

fonksiyon Sayaç: herhangi
değişken
  n: tamsayı
başla
  n := 0
  dön fonksiyon başla n := n + 1; sonuç := n; bitir
```

```
bitir

değişken s: herhangi

başla
  sayılar := [1..10]
  kare := fonksiyon(x) => x * x            // isimsiz fonksiyon (lambda)
  yaz('kare(7)  = ' + Çağır(kare, 7), 1)

  kareler := Eşle(sayılar, kare)
  çiftler := Süz(sayılar, fonksiyon(x) => x mod 2 = 0)
  genelToplam := İndirge(sayılar, fonksiyon(a, b) => a + b)
  yaz('Kareler  : ' + kareler, 1)
  yaz('Çiftler  : ' + çiftler, 1)
  yaz('Toplam   : ' + genelToplam, 1)

  // boru işleci: soldaki değer, sağdaki fonksiyonun ilk argümanı olur
  yaz('Boru     : ' + (sayılar |> Süz(fonksiyon(x) => x > 5) |> Eşle(kare) |>
      ↪ Toplam), 1)

  kişiler := [['ad': 'Can', 'yaş': 31], ['ad': 'Ada', 'yaş': 24], ['ad': 'Ege',
      ↪ 'yaş': 28]]
  yaz('Yaşa göre: ' + Eşle(Sırala(kişiler, fonksiyon(k) => k['yaş']), fonksiyon(k) =>
      ↪ k['ad']), 1)

  s := Sayaç                               // kapanış: n değişkenini hatırlar
  Çağır(s);  Çağır(s)
  yaz('Sayaç    : ' + Çağır(s), 1)
bitir.
```

```
kare(7)  = 49
Kareler  : [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Çiftler  : [2, 4, 6, 8, 10]
Toplam   : 55
Boru     : 330
Yaşa göre: ['Ada', 'Ege', 'Can']
Sayaç    : 3
```

Araç	Ne yapar?
`fonksiyon(x) => ifade`	Tek ifadelik isimsiz fonksiyon
`fonksiyon(x) başla … bitir`	Gövdeli isimsiz fonksiyon
`Eşle(L, f)`	Her elemana `f` uygular, yeni liste döndürür
`Süz(L, f)`	`f` doğru döndüren elemanları tutar
`İndirge(L, f)`	Elemanları ikili ikili birleştirerek tek değere indirir

Araç	Ne yapar?
`Sırala(L, anahtar)`	Anahtar fonksiyonunun sonucuna göre sıralar
`x |> f(y)`	`f(x, y)` ile aynıdır; işlem zincirleri soldan sağa okunur
`Çağır(f, ...)`	Değişkende duran bir fonksiyonu çağırır
`@Ad`	Adı verilen fonksiyonu çağırmadan değer olarak alır

Kapanış (closure): `Sayaç` içinde üretilen lambda, `n` yerel değişkenini "hatırlar". Her çağrıda aynı `n` artar.

Dikkat: Bu örnekte değişkene `toplam` yerine `genelToplam` dedik. Yöntem büyük/küçük harf ayırmadığı için `toplam` adı yerleşik `Toplam` fonksiyonunu gölgelerdi; boru zincirindeki `|> Toplam` çalışmazdı. Bu nedenle değişkenlere yerleşik işlevlerin adlarını vermemek gerekir.

Alıştırmalar

1. Bir metindeki ünlü harfleri (a, e, ı, i, o, ö, u, ü) sayan `ÜnlüSayısı(s: metin): tamsayı` fonksiyonunu yazınız.
2. Fibonacci sayılarının ilk 15'ini bir listede toplayınız; sonra boru işleciyle yalnızca çiftleri süzüp toplamını yazdırınız.
3. Öğrenci adlarını ve notlarını bir sözlükte tutunuz; notu 70'in üzerinde olanları alfabetik sırayla yazdırınız.

BÖLÜM

9

Kayıtlar ve sınıflar

Gerçek dünyadaki şeylerin birden çok özelliği vardır: bir noktanın x ve y'si, bir hesabın sahibi ve bakiyesi. **Kayıt** bu özellikleri bir araya toplar. **Sınıf** ise özelliklerin yanına davranışları (metodları) da ekler.

Kayıtlar

```
program KayıtÖrneği
tip
  Nokta = kayıt
    x, y: ondalık
  bitir

fonksiyon Uzaklık(a, b: Nokta): ondalık
başla
  dön Kök((a.x - b.x) ** 2 + (a.y - b.y) ** 2)
bitir
```

```
değişken
  p, q: Nokta
başla
  p.x := 1;  p.y := 2
  q := Nokta.Oluştur(4, 6)      // alanları sırayla doldurur
  yaz('p = ' + p + ', q = ' + q, 1)
  yaz('Uzaklık: ' + Uzaklık(p, q), 1)
  q := p                        // kayıtlar KOPYALANIR
  q.x := 99
  yaz('p.x hâlâ ' + p.x, 1)
bitir.
```

```
p = Nokta(x: 1.0, y: 2.0), q = Nokta(x: 4.0, y: 6.0)
Uzaklık: 5.0
p.x hâlâ 1.0
```

Kayıt tipinde bildirilen bir değişken boş alanlarla hazır gelir; `p.x := 1` hemen yazılabilir. Kayıtlar **değer gibi** davranır: `q := p` bir kopya üretir, `q`'yu değiştirmek `p`'yi etkilemez.

Sınıflar

Bir sınıf iki parçada yazılır. Önce `tip` bölümünde sınıfın neye sahip olduğu bildirilir. Sonra metodların gövdeleri `Sınıf.Metod` adıyla tanımlanır. Bu Delphi tarzıdır ve büyük sınıfları okunur tutar.

```
program BankaHesabı
tip
  Hesap = sınıf
  özel
    tutar: ondalık
  genel
    sahip: metin
    kurucu Oluştur(ad: metin; ilkPara: ondalık = 0)
    prosedür Yatır(miktar: ondalık)
    prosedür Çek(miktar: ondalık)
    fonksiyon Bakiye: ondalık
    fonksiyon MetneÇevir: metin
  bitir

kurucu Hesap.Oluştur(ad: metin; ilkPara: ondalık = 0)
başla
  sahip := ad
  tutar := ilkPara
```

```
bitir

prosedür Hesap.Yatır(miktar: ondalık)
başla
  tutar := tutar + miktar
bitir

prosedür Hesap.Çek(miktar: ondalık)
başla
  eğer miktar > tutar ise
    fırlat Hata('Yetersiz bakiye', 'BankaHatası')
  tutar := tutar - miktar
bitir

fonksiyon Hesap.Bakiye: ondalık
başla
  dön tutar
bitir

fonksiyon Hesap.MetneÇevir: metin
başla
  dön Biçimle('{} hesabı: {:.2} TL', sahip, tutar)
bitir

değişken
  h: Hesap
başla
  h := Hesap.Oluştur('EdaEce', 100)
  h.Yatır(250)
  h.Çek(80)
  yaz(h, 1)
  yaz('Bakiye: ' + h.Bakiye, 1)
  dene
    h.Çek(1000)
  yakala (hata)
    yaz(hata.tür + ': ' + hata.mesaj, 1)
  bitir
bitir.
```

```
EdaEce hesabı: 270.00 TL
Bakiye: 270.0
BankaHatası: Yetersiz bakiye
```

Kavram	Açıklama
Alan	Sınıfın verisi: `tutar`, `sahip`. Metodlar içinden doğrudan adıyla kullanılır.

Kavram	Açıklama
Metod	Sınıfa ait prosedür ya da fonksiyon: `Yatır`, `Çek`
`kurucu`	Yeni nesne oluşturan özel metod. `Hesap.Oluştur(...)` ile çağrılır.
`kendi`	Metodun üzerinde çalıştığı nesne. `kendi.tutar = tutar`
`özel` / `genel`	Hangi üyelerin dışarı için olduğunu belgeler. Şimdilik bilgi amaçlıdır, erişimi engellemez.
`MetneÇevir`	Tanımlanırsa `yaz(nesne)` onu kullanır

Alanlara ve metodlara aynı adı vermemek gerekir. Büyük/küçük harf fark etmediği için `bakiye` alanı ile `Bakiye` metodu aynı ad olur. Bu yüzden alana `tutar` dedik.

Kalıtım

Bir sınıf başka bir sınıfın özelliklerini **miras alabilir** ve istediği metodları değiştirebilir. Metod gövdeleri sınıfın içinde de yazılabilir; kısa sınıflar için bu daha pratiktir.

```
program Kalıtım
tip
  Hayvan = sınıf
    ad: metin
    kurucu Oluştur(a: metin)
    başla
      ad := a
    bitir
    fonksiyon Ses: metin; sanal
    başla
      dön '...'
    bitir
    fonksiyon Tanıt: metin
    başla
      dön ad + ' der ki: ' + Ses
    bitir
  bitir

  Kedi = sınıf(Hayvan)
    fonksiyon Ses: metin; ezer
    başla dön 'Miyav'; bitir
  bitir
```

```
  Köpek = sınıf(Hayvan)
    cins: metin
    kurucu Oluştur(a, c: metin)
    başla
      miras Oluştur(a)            // ebeveynin kurucusunu çağır
      cins := c
    bitir
    fonksiyon Ses: metin; ezer
    başla dön 'Hav hav'; bitir
    fonksiyon Tanıt: metin; ezer
    başla dön miras Tanıt + ' (' + cins + ')'; bitir
  bitir

değişken
  hayvanlar: liste
  h: Hayvan
başla
  hayvanlar := [Kedi.Oluştur('Tekir'), Köpek.Oluştur('Karabaş', 'Kangal'),
      ↪  Hayvan.Oluştur('Balık')]
  hayvanlar içindeki her h için
    yaz(h.Tanıt, 1)
  yaz('Karabaş bir Hayvan mı? ' + ÖrneğiMi(hayvanlar[1], Hayvan), 1)
  yaz('Tekir bir Köpek mi?    ' + ÖrneğiMi(hayvanlar[0], Köpek), 1)
bitir.
```

```
Tekir der ki: Miyav
Karabaş der ki: Hav hav (Kangal)
Balık der ki: ...
Karabaş bir Hayvan mı? doğru
Tekir bir Köpek mi?    yanlış
```

Buradaki en önemli fikir **çok biçimliliktir**. `Tanıt` metodu yalnızca `Hayvan` sınıfında yazıldı, ama içindeki `Ses` çağrısı nesnenin gerçek sınıfına göre "Miyav" ya da "Hav hav" oldu. `miras` ebeveyn sınıfın aynı adlı metodunu çağırır. `sanal` ve `ezer` sözcükleri okuyana niyeti anlatır; Yöntem'de tüm metodlar zaten sanaldır.

BÖLÜM

10

Hatalarla başa çıkmak

Programlar beklenmedik durumlarla karşılaşır: sıfıra bölme, olmayan bir dosya, hatalı girdi. Yöntem bu durumlarda bir **hata fırlatır**. Hatayı `dene … yakala … sonunda … bitir` ile yakalayıp programı durdurmadan yönetebilirsiniz.

```
program HataYönetimi
değişken
  sayılar: liste
  i, bölen: tamsayı

fonksiyon Bölüştür(toplam, kişi: tamsayı): tamsayı
başla
  eğer kişi < = 0 ise
    fırlat Hata('Kişi sayısı pozitif olmalı', 'GeçersizGirdi')
  dön toplam böl kişi
bitir

başla
```

```
  sayılar := [10, 20, 30]
  dene
    yaz(sayılar[5], 1)
  yakala (h)
    yaz('Yakalandı: ' + h.mesaj, 1)
  bitir

  [3, 0, -2] içindeki her bölen için
  başla
    dene
      yaz('100 / ' + bölen + ' kişi = ' + Bölüştür(100, bölen), 1)
    yakala (h)
      yaz(h.tür + ': ' + h.mesaj, 1)
    sonunda
      yaz('  (hesap denendi)', 1)
    bitir
  bitir

  İddia(Uzunluk(sayılar) = 3, 'Liste 3 elemanlı olmalı')
  yaz('İddia doğrulandı.', 1)
bitir.
```

```
Yakalandı: Liste dizini sınır dışı: 5 (uzunluk 3)
100 / 3 kişi = 33
  (hesap denendi)
GeçersizGirdi: Kişi sayısı pozitif olmalı
  (hesap denendi)
GeçersizGirdi: Kişi sayısı pozitif olmalı
  (hesap denendi)
İddia doğrulandı.
```

Yapı	Görevi
`dene …`	Hata çıkabilecek kodu çevreler
`yakala (h) …`	Hata olursa çalışır. `h.mesaj`, `h.tür`, `h.satır` bilgileri vardır. `(h)` isteğe bağlıdır.
`sonunda …`	Hata olsa da olmasa da her zaman çalışır; dosya kapatma gibi temizlik işleri için
`fırlat Hata(mesaj, tür)`	Kendi hatanızı fırlatır
`İddia(koşul, mesaj)`	Koşul yanlışsa hata fırlatır; testler için idealdir

Yakalanmayan bir hata programı durdurur ve hangi dosyanın hangi satırında olduğunu yazar. Bu mesajları okumayı alışkanlık haline getirmenizi öneririz: mesajlar, hatanın ne olduğunu ve nerede olduğunu genellikle tam olarak söyler.

BÖLÜM

11

Birimler: programı parçalara bölmek

Büyüyen bir programı tek dosyada tutmak zorlaşır. **Birim**, başka programların kullanabileceği bağımsız bir kod dosyasıdır. Bir birimin iki yüzü vardır: `arayüz` dışarıya ne sunduğunu, `uygulama` ise bunu nasıl yaptığını içerir.

`Geometri.y` dosyası şöyledir:

```
birim Geometri
{ Basit geometri hesapları }

arayüz                                  // dışarıya açılan adlar
  sabit BİRİM_ADI: metin = 'cm'
  fonksiyon DaireAlanı(r: ondalık): ondalık
  fonksiyon DikdörtgenAlanı(en, boy: ondalık): ondalık

uygulama                                // ayrıntılar burada gizli
  kullan Matematik

  değişken çağrıSayısı: tamsayı
```

```
  fonksiyon DaireAlanı(r: ondalık): ondalık
  başla
    çağrıSayısı + = 1
    dön pi * Üs(r, 2)
  bitir

  fonksiyon DikdörtgenAlanı(en, boy: ondalık): ondalık
  başla
    çağrıSayısı + = 1
    dön en * boy
  bitir

başla                                  // birim ilk yüklendiğinde çalışır
  çağrıSayısı := 0
  yaz('[Geometri birimi yüklendi]', 1)
bitir.
```

Aynı klasördeki `ana.y` dosyası ise şöyledir:

```
program AlanHesabı
kullan Geometri
başla
  yaz(Biçimle('Daire (r =2)      : {:.2} {}²', DaireAlanı(2), BİRİM_ADI), 1)
  yaz(Biçimle('Dikdörtgen (3x4) : {:.2} {}²', Geometri.DikdörtgenAlanı(3, 4),
      ↪ BİRİM_ADI), 1)
  yaz(Geometri.çağrıSayısı, 1)
bitir.
```

```
[Geometri birimi yüklendi]
Daire (r =2)      : 12.57 cm²
Dikdörtgen (3x4) : 12.00 cm²
Çalışma hatası [ana.y, satır 6] 'Geometri' biriminde 'cagrisayisi' dışa aktarılmamış
```

Son satırdaki hata bilerek bırakıldı: `çağrıSayısı` yalnızca `uygulama` bölümünde tanımlıdır; bu yüzden dışarıdan görünmez. Birimler iç ayrıntıları böyle korur.

- `kullan Geometri;` birimi yükler. Dışa açılan adlar doğrudan (`DaireAlanı`) ya da birim adıyla (`Geometri.DikdörtgenAlanı`) kullanılır.
- Birimin `başla … bitir.` bölümü, birim ilk kez yüklendiğinde bir kez çalışır.
- Birim dosyaları şu sırayla aranır: programın klasörü, onun `kutuphane` alt klasörü, Yöntem'in kendi `kutuphane` klasörü.

Yöntem'le birlikte gelen hazır birimler şunlardır:

Birim	İçerik
`Matematik`	`Sin`, `Cos`, `Ln`, `Üs`, `Taban`, `Tavan`, `Faktöriyel`, `pi`, `e`…
`Sistem`	`Zaman`, `Tarih`, `DosyaOku`, `DosyaYaz`, `Argümanlar`…
`Veri`	`CSVOku`, `CSVYaz`, `JSONÇöz`, `JSONYap`, `Standartlaştır`…
`İstatistik`	`Ortalama`, `Medyan`, `StandartSapma`, `Korelasyon`…
`YZ`	Tensörler, otomatik türev, sinir ağları (Bölüm 12--13)
`Arama`, `Uzman`, `Dil`, `MakineÖğrenmesi`	Klasik yapay zekâ (Bölüm 14)
`Kuantum`, `KuantumAlgoritmaları`	Kuantum programlama (Bölüm 15)

Alıştırmalar

1. `Dikdörtgen` adında bir sınıf yazınız: `en`, `boy` alanları, `Alan` ve `Çevre` metodları olsun. Ondan türeyen ve kurucusu tek kenar alan `Kare` sınıfını `miras` ile yazınız.
2. `Hesap` sınıfına, her işlemi bir listeye kaydeden ve `Geçmiş` metoduyla bu listeyi döndüren bir özellik ekleyiniz.
3. Alıştırma 1'deki sınıfları `Şekiller` adında bir birime taşıyınız ve başka bir programdan kullanınız.

BÖLÜM

12

Tensörler ve otomatik türev

Modern yapay zekânın temelinde iki fikir vardır: verileri sayı tabloları (**tensörler**) olarak işlemek ve bir hatayı azaltmak için ayarları hangi yöne değiştireceğimizi gösteren **türevi** hesaplamak. Yöntem'in `YZ` birimi ikisini de dilin içine yerleştirir.

Tensörlerle işlem

```
program Tensörler
kullan YZ
değişken
  A, B, v: tensör
başla
  A := Tensör([[1, 2], [3, 4]])
  B := Tensör([[0, 1], [1, 0]])
  v := Tensör([10, 20])
  yaz('A =', 1);  yaz(A, 1)
  yaz('Şekil       : ' + A.Şekil, 1)
```

```
  yaz('A + v       : ' + (A + v), 1)      // yayınlama: v her satıra eklenir
  yaz('A * 2       : ' + (A * 2), 1)
  yaz('A @ B       : ' + (A @ B), 1)      // matris çarpımı
  yaz('A.T         : ' + A.T, 1)          // devrik
  yaz('Toplam      : ' + A.Toplam.Değer, 1)
  yaz('Satır topl. : ' + A.Toplam(1), 1)
  yaz('A[1, 0]     : ' + A[1, 0], 1)
  yaz('Sigmoid(0)  : ' + Sigmoid(Tensör(0.0)), 1)
bitir.
```

```
A =
tensör([[1, 2],
        [3, 4]])
Şekil       : [2, 2]
A + v       : tensör([[11, 22],
        [13, 24]])
A * 2       : tensör([[2, 4],
        [6, 8]])
A @ B       : tensör([[2, 1],
        [4, 3]])
A.T         : tensör([[1, 3],
        [2, 4]])
Toplam      : 10.0
Satır topl. : tensör([3, 7])
A[1, 0]     : 3.0
Sigmoid(0)  : tensör(0.5)
```

- `*` öğe öğe çarpar; `@` matris çarpımı yapar.
- **Yayınlama:** Şekilleri farklı tensörler uyumluysa küçüğü genişletilir. `A + v`'de `[10, 20]` her satıra eklendi.
- `Toplam`, `Ortalama` bütün tensör için ya da bir eksen boyunca (`0` sütunlar, `1` satırlar) çalışır. Tek değerli bir tensörü sayıya çevirmek için `.Değer` kullanılır.

Otomatik türev ve gradyan inişi

Bir tensörü `Tensör(değer, doğru)` ile oluşturursanız, Yöntem onunla yapılan her işlemi kaydeder. Sonuç üzerinde `GeriYay` çağırınca tüm türevler zincir kuralıyla hesaplanır.

```
program OtomatikTürev
kullan YZ
değişken
  x, y: tensör
  w: tensör
  i: tamsayı
```

```
  f: herhangi
başla
  // 1) Elle: y = x² + 3x, x = 2 noktasında dy/dx = 2x + 3 = 7
  x := Tensör(2.0, doğru)           // doğru: gradyanı izle
  y := x * x + 3 * x
  y.GeriYay                          // gradyanları hesapla
  yaz('y = ' + y.Değer + ',  dy/dx = ' + x.Grad.Değer, 1)

  // 2) Türev fonksiyonuyla: herhangi bir Yöntem fonksiyonunun türevi
  f := fonksiyon(t) => t ** 3 - 2 * t
  yaz('f''(1.5) = ' + Türev(f, 1.5), 1)  // 3t² - 2 = 4.75

  // 3) Gradyan inişi: g(w) = (w - 5)² fonksiyonunun en küçük noktasını bul
  w := Tensör(0.0, doğru)
  1'den 50'ye kadar her i için
  başla
    y := (w - 5) ** 2
    w.SıfırlaGrad
    y.GeriYay
    w := (w - 0.1 * w.Grad).Ayır     // gradyanın tersi yönünde küçük bir adım
    w.GradGerekli := doğru
    eğer i mod 10 = 0 ise
      yaz(Biçimle('  adım {:2}: w = {:.4}', i, w.Değer), 1)
  bitir
bitir.
```

```
y = 10.0,  dy/dx = 7.0
f'(1.5) = 4.75
  adım 10: w = 4.4631
  adım 20: w = 4.9424
  adım 30: w = 4.9938
  adım 40: w = 4.9993
  adım 50: w = 4.9999
```

Üçüncü parça, makine öğreniminin kalbindeki **gradyan inişi**dir. Türev, fonksiyonun hangi yönde arttığını gösterir; biz ters yönde küçük adımlar atarız. `w` her adımda en küçük noktaya, 5'e yaklaşır. Bir sinir ağını eğitmek de tam olarak budur; yalnızca binlerce `w` vardır.

- `.Ayır` güncellenen değeri eski hesaplama grafiğinden koparan yeni bir tensör üretir.
- `SıfırlaGrad` önceki adımın gradyanını temizler; yoksa gradyanlar birikir.

BÖLÜM

13

Sinir ağları

Bir sinir ağı, katmanların art arda dizilmesiyle oluşur. Her **Yoğun** katman girdisini bir ağırlık matrisiyle çarpar, bir sapma ekler ve sonucu bir **etkinleştirme fonksiyonundan** (`relu`, `tanh`, `sigmoid`) geçirir. Eğitim, bu ağırlıkları gradyan inişiyle ayarlamaktır.

İlk sinir ağı: XOR

```
program SinirAğı
kullan YZ
değişken
  X, Y, tahmin: tensör
  model, opt: herhangi
  geçmiş: liste
başla
  Tohum(42)
  // XOR: girdiler aynıysa 0, farklıysa 1. Tek bir doğruyla ayrılamaz.
```

```
  X := Tensör([[0, 0], [0, 1], [1, 0], [1, 1]])
  Y := Tensör([[0], [1], [1], [0]])

  model := Ardışık([Yoğun(2, 8, 'tanh'),       // 2 girdi -> 8 gizli nöron
                    Yoğun(8, 1, 'sigmoid')])  // 8 -> 1 çıktı (0..1 olasılık)
  opt := Adam(model.Parametreler, 0.05)
  yaz('Parametre sayısı: ' + model.ParametreSayısı, 1)

  geçmiş := Eğit(model, X, Y, 'ikiliÇaprazEntropi', opt, 300, 100)

  tahmin := model.Tahmin(X)
  yaz('Tahminler:', 1)
  yaz(tahmin, 1)
  yaz('Doğruluk : %' + Doğruluk(tahmin, [0, 1, 1, 0]) * 100, 1)
bitir.
```

```
Parametre sayısı: 33
  Epok     1/300   kayıp = 0.704415
  Epok   100/300   kayıp = 0.001678
  Epok   200/300   kayıp = 0.000737
  Epok   300/300   kayıp = 0.000424
Tahminler:
tensör([[0.0002],
        [0.9997],
        [0.9993],
        [0.0005]])
Doğruluk : %100.0
```

Parça	Görevi
`Ardışık([...])`	Katmanları sırayla birbirine bağlayan model
`Yoğun(g, ç, etkinleştirme)`	`g` girdi alan, `ç` çıktı veren tam bağlantılı katman
`Adam(parametreler, oran)`	Ağırlıkları güncelleyen optimizasyon algoritması. Basit olanı: `SGD`
`Eğit(model, X, Y, kayıp, opt, epok, rapor)`	Hazır eğitim döngüsü; kayıp geçmişini döndürür
Kayıp fonksiyonları	`OKH` (sayı tahmini), `İkiliÇaprazEntropi` (evet/hayır), `ÇaprazEntropi` (çok sınıf)
`Tahmin(X)`	Eğitimden sonra modeli kullanır

Kayıp modelin ne kadar yanıldığını ölçer. 0.70'ten 0.0004'e düşmesi, ağın XOR kuralını öğrendiğini gösterir. `Tohum(42)` başlangıç ağırlıklarını sabitler; böylece sonuçlar her çalıştırmada aynı çıkar.

Kendi eğitim döngünüz: üç sınıflı sınıflandırma

`Eğit` kolaydır ama ne olduğunu görmek için döngüyü bir kez elle yazmak öğreticidir. Aşağıdaki program üç nokta bulutu üretir ve hangi noktanın hangi buluta ait olduğunu öğrenir.

```
program ÜçSınıf
kullan YZ, Matematik
değişken
  noktalar: liste
  etiketler: liste
  i, s: tamsayı
  X: tensör
  model, opt: herhangi
  kayıp: tensör
  epok: tamsayı
  merkezler: liste
başla
  Tohum(1)
  merkezler := [[0, 0], [3, 0], [1.5, 3]]
  noktalar := [];  etiketler := []
  0'dan 2'ye kadar her s için
    1'den 30'a kadar her i için
    başla
      noktalar.Ekle([merkezler[s][0] + Normal(0, 0.6), merkezler[s][1] + Normal(0,
          ↪ 0.6)])
      etiketler.Ekle(s)
    bitir
  X := Tensör(noktalar)

  model := Ardışık([Yoğun(2, 16, 'relu'), Yoğun(16, 3)])
  opt := Adam(model.Parametreler, 0.03)
  1'den 150'ye kadar her epok için
  başla
    opt.SıfırlaGrad                                    // 1. eski gradyanları sil
    kayıp := ÇaprazEntropi(model.İleri(X), etiketler)  // 2. ileri geçiş + kayıp
    kayıp.GeriYay                                      // 3. gradyanları hesapla
    opt.Adım                                           // 4. ağırlıkları güncelle
    eğer epok mod 50 = 0 ise
      yaz(Biçimle('epok {:3}  kayıp {:.4}  doğruluk %{:.1}', epok, kayıp.Değer,
                  Doğruluk(model.Tahmin(X), etiketler) * 100), 1)
  bitir
  yaz('Yeni nokta [2.8, 0.2] sınıfı: ' + ArgMaks(model.Tahmin([[2.8, 0.2]]))[0], 1)
```

```
  yaz('Yeni nokta [1.4, 2.7] sınıfı: ' + ArgMaks(model.Tahmin([[1.4, 2.7]]))[0], 1)
bitir.
```

```
epok  50  kayıp 0.0225  doğruluk %100.0
epok 100  kayıp 0.0137  doğruluk %100.0
epok 150  kayıp 0.0109  doğruluk %100.0
Yeni nokta [2.8, 0.2] sınıfı: 1
Yeni nokta [1.4, 2.7] sınıfı: 2
```

Her sinir ağı eğitimi bu dört adımı tekrarlar: **sıfırla → ileri geç ve kaybı hesapla → geri yay → güncelle.** Son katmanda etkinleştirme olmaması bilinçlidir: `ÇaprazEntropi` ham puanları kendisi olasılığa çevirir. `ArgMaks` en yüksek puanlı sınıfın numarasını verir. (3, 0) civarındaki nokta 1. buluta, (1.5, 3) civarındaki 2. buluta atandı.

Kendi katmanınızı yazmak: `İleri` ve `Parametreler` metodları olan her Yöntem sınıfı `Ardışık` modele katman olarak eklenebilir. Örneği Yöntem klasöründeki `ornekler/xor_sinir_agi.y` dosyasında bulabilirsiniz.

BÖLÜM

14

Klasik yapay zekâ: arama, uzman sistemler, dil işleme

Yapay zekâ yalnızca sinir ağlarından ibaret değildir. Yol bulma, kurallarla akıl yürütme ve metin sınıflandırma da yapay zekânın klasik konularıdır. Bu birimlerin hepsi Yöntem dilinde yazılmıştır; `kutuphane` klasöründe kodlarını okuyabilirsiniz.

Arama algoritmaları

```
program Labirent
kullan Arama
değişken
  metro: sözlük
  harita, yol: liste
başla
  // Çizge: her durağın komşuları
  metro := ['Taksim': ['Şişhane', 'Osmanbey'],
            'Şişhane': ['Haliç', 'Taksim'],
            'Haliç': ['Vezneciler', 'Şişhane'],
```

```
            'Osmanbey': ['Şişli', 'Taksim'],
            'Şişli': ['Gayrettepe'],
            'Vezneciler': ['Yenikapı']]
  yaz('Taksim -> Yenikapı: ' + GenişlikÖncelikli(metro, 'Taksim', 'Yenikapı'), 2)

  // A*: '#' duvar, '.' yol
  harita := ['.....#....',
             '.###.#.##.',
             '...#...#..',
             '##.#####.#',
             '..........']
  yol := AYıldız(harita, [0, 0], [0, 9])
  yaz('A* ile ' + (Uzunluk(yol) - 1) + ' adımda:', 1)
  yaz(IzgaraÇiz(harita, yol), 1)
bitir.
```

```
Taksim -> Yenikapı: ['Taksim', 'Şişhane', 'Haliç', 'Vezneciler', 'Yenikapı']

A* ile 13 adımda:
*****#****
.###*#*##.
...#***#..
##.#####.#
..........
```

Genişlik öncelikli arama başlangıçtan dışa doğru katman katman ilerler ve en az duraklı yolu bulur. **A*** ise her adımda hedefe kalan tahmini uzaklığı da hesaba katar; bu sayede gereksiz yerlere sapmadan en kısa yolu bulur. Yıldızlar bulunan yolu gösterir.

Uzman sistem

Uzman sistem, bir alandaki bilgiyi **EĞER ... İSE ...** kuralları olarak saklar ve bilinen olgulardan yeni sonuçlar çıkarır.

```
program HavaDanışmanı
kullan Uzman
değişken
  bt: BilgiTabanı
  adım: herhangi
başla
  bt := BilgiTabanı.Oluştur
  bt.KuralEkle('K1', ['bulutlu', 'nem yüksek'], 'yağmur olası', 0.8)
  bt.KuralEkle('K2', ['yağmur olası'], 'şemsiye al', 0.9)
  bt.KuralEkle('K3', ['güneşli', 'sıcak'], 'şapka tak', 1.0)
```

```
  bt.OlguEkle('bulutlu')
  bt.OlguEkle('nem yüksek', 0.9)

  bt.İleriZincirle içindeki her adım için
    yaz('Ateşlendi: ' + adım, 1)
  yaz(Biçimle('Şemsiye önerisinin güveni: {:.2}', bt.Güven('şemsiye al')), 1)
  yaz('Neden? ' + bt.Açıkla('şemsiye al'), 1)
  yaz('Şapka gerekli mi? ' + bt.GeriZincirle('şapka tak'), 1)
bitir.
```

```
Ateşlendi: K1 => yağmur olası
Ateşlendi: K2 => şemsiye al
Şemsiye önerisinin güveni: 0.80
Neden? şemsiye al <- K2 [yağmur olası <- K1 [bulutlu (verilen olgu); nem yüksek (verilen
    ↪ olgu)]]
Şapka gerekli mi? yanlış
```

- **İleri zincirleme** olgulardan başlar ve yeni olgu üretilemeyene dek kuralları ateşler.
- **Geri zincirleme** bir hedeften başlar ve onu kanıtlayacak olguları arar.
- Her kuralın bir **güven** katsayısı vardır. Bir sonucun güveni, onu destekleyen zincirdeki en zayıf halkadır.
- `Açıkla` sistemin neden o sonuca vardığını gösterir. Açıklanabilirlik, uzman sistemlerin güçlü yanıdır.

Metin sınıflandırma: spam filtresi

```
program SpamFiltresi
kullan Dil
değişken
  filtre: NaifBayes
  m: herhangi
başla
  filtre := NaifBayes.Oluştur
  filtre.Eğit(
    ['Hemen tıkla, büyük ödül kazandın',
     'Bedava hediye çeki seni bekliyor, tıkla',
     'Son fırsat! Ücretsiz kazan',
     'Yarınki toplantı saat onda',
     'Raporu akşama kadar gönderebilir misin',
     'Toplantı notlarını ekte bulabilirsin'],
    ['spam', 'spam', 'spam', 'normal', 'normal', 'normal'])
```

```
  ['Ücretsiz ödül için tıkla', 'Toplantı raporu ekte', 'Bedava hediye kazan']
      ↪ içindeki her m için
    yaz(m + '  ->  ' + filtre.Tahmin(m), 1)
  yaz('Belirteçler: ' + Belirteçle('Yapay zekâ, dili anlamaya çalışıyor!'), 1)
bitir.
```

```
Ücretsiz ödül için tıkla  ->  spam
Toplantı raporu ekte  ->  normal
Bedava hediye kazan  ->  spam
Belirteçler: ['yapay', 'zekâ', 'dili', 'anlamaya', 'çalışıyor']
```

Naif Bayes, her sözcüğün her sınıfta ne sıklıkla geçtiğini öğrenir. Yeni bir metinde, sözcüklerin hangi sınıfı daha olası kıldığını hesaplar. Yalnızca altı örnekle bile "ücretsiz", "tıkla", "hediye" gibi sözcüklerin spam'e işaret ettiğini öğrendi.

Alıştırmalar

1. `OKH` kaybını kullanarak $y = 2x^2 - 1$ eğrisine uyan bir sinir ağı eğitiniz. x için -1 ile 1 arasından 20 nokta alabilirsiniz.
2. Hava danışmanına üç yeni kural ekleyiniz ve `GeriZincirle` ile bir sonucu sorgulayınız.
3. Spam filtresini kendi örneklerinizle eğitiniz; `Olasılıklar` metoduyla her sınıfın olasılığını yazdırınız.

BÖLÜM

15

Kuantum programlama

Klasik bir bit ya 0 ya 1'dir. Bir **kübit** ise ölçülene kadar 0 ile 1'in bir **süperpozisyonunda** bulunabilir. Ölçtüğünüzde belirli olasılıklarla 0 ya da 1 olur. Yöntem'in `Kuantum` birimi, bilgisayarınızda en çok 20 kübitlik kuantum devreleri kurup simüle etmenizi sağlar.

İlk kübit

```
program İlkKübit
kullan Kuantum
değişken
  d: herhangi
  i, birler: tamsayı
başla
  KuantumTohum(5)
  d := Devre(1)                       // bir kübit, |0⟩ durumunda
  yaz('Başlangıç : ' + d.Durum, 1)
```

```
  d.X(0)                              // NOT kapısı: |0⟩ -> |1⟩
  yaz('X sonrası : ' + d.Durum, 1)
  d.H(0)                              // Hadamard: süperpozisyon
  yaz('H sonrası : ' + d.Durum, 1)
  yaz('Olasılıklar: ' + d.Olasılıklar, 2)

  // 100 kez hazırla ve ölç
  birler := 0
  1'den 100'e kadar her i için
  başla
    d := Devre(1)
    d.H(0)
    birler + = d.Ölç(0)
  bitir
  yaz('100 ölçümde 1 çıkma sayısı: ' + birler, 1)
bitir.
```

```
Başlangıç : 1|0⟩
X sonrası : 1|1⟩
H sonrası : 0.7071|0⟩ - 0.7071|1⟩
Olasılıklar: ['0': 0.5, '1': 0.5]

100 ölçümde 1 çıkma sayısı: 50
```

`|0⟩` ve `|1⟩` ("ket sıfır", "ket bir") kübitin temel durumlarıdır. Hadamard kapısından sonra kübit ikisinin karışımı oldu. Genliklerin karesi olasılığı verir: $0.7071^2 \approx 0.5$. Eksi işareti olasılığı değiştirmez ama girişimde (interference) önemlidir. `Ölç` durumu çökertir ve 0 ya da 1 döndürür; 100 denemenin yarısı 1 çıktı.

Dolanıklık

İki kübit **dolanık** hale getirildiğinde, birini ölçmek diğerinin sonucunu da belirler.

```
program Dolanıklık
kullan Kuantum
değişken
  d: herhangi
  a, b, i: tamsayı
başla
  KuantumTohum(3)
  d := Devre(2)
  d.H(0)
  d.CX(0, 1)                          // q0 1 ise q1'i çevir
  yaz(d.Çiz, 1)
  yaz('Durum: ' + d.Durum, 1)
```

```
  yaz('1000 atış: ' + d.Çalıştır(1000), 2)

  1'den 5'e kadar her i için
  başla
    d := Devre(2)
    d.H(0);  d.CX(0, 1)
    a := d.Ölç(0)
    b := d.Ölç(1)
    yaz('q0 = ' + a + ', q1 = ' + b, 1)
  bitir
bitir.
```

```
q0: ─H─●──
q1: ───X──
Durum: 0.7071|00⟩ + 0.7071|11⟩
1000 atış: ['00': 509, '11': 491]

q0 = 1, q1 = 1
q0 = 0, q1 = 0
q0 = 0, q1 = 0
q0 = 1, q1 = 1
q0 = 0, q1 = 0
```

Bu, ünlü **Bell durumudur**. Sonuçların her biri tek başına rastgeledir, ama iki kübit her zaman aynı değeri verir; `01` ya da `10` hiç çıkmaz. `Çiz` devreyi metin olarak gösterir: ● kontrol kübiti, `X` hedeftir. `Çalıştır(1000)` devreyi 1000 kez çalıştırıp sonuçları sayar.

Bit sırası: Yöntem'de bit dizilerinde q0 en soldadır. `'10'`, q0 = 1 ve q1 = 0 demektir.

Temel kapılar

Kapı	Yöntem	Etkisi
X (NOT)	`d.X(q)`	$0 \leftrightarrow 1$
Hadamard	`d.H(q)`	Eşit süperpozisyon oluşturur
Z, S, T	`d.Z(q)`, `d.S(q)`, `d.T(q)`	Fazı değiştirir; olasılıklar aynı kalır
Dönmeler	`d.Rx(q, θ)`, `d.Ry(q, θ)`, `d.Rz(q, θ)`	Kübiti bir eksen etrafında θ kadar döndürür
CNOT	`d.CX(k, h)`	Kontrol 1 ise hedefi çevirir

Kapı	Yöntem	Etkisi
Toffoli	`d.CCX(k1, k2, h)`	İki kontrol de 1 ise hedefi çevirir
Takas	`d.Takas(a, b)`	İki kübiti yer değiştirir
Ölçüm	`d.Ölç(q)`	0 ya da 1 döndürür, durum çöker

Kuantum araması: Grover algoritması

Sırasız bir listede bir elemanı bulmak için klasik bir bilgisayar ortalama $N/2$ deneme yapar. Grover algoritması bunu yaklaşık $\sqrt{N}$ adımda yapar. `KuantumAlgoritmaları` birimi algoritmayı hazır sunar.

```
program GroverArama
kullan Kuantum, KuantumAlgoritmaları
değişken
  d: herhangi
  sayım: sözlük
başla
  KuantumTohum(11)
  // 3 kübit = 8 olasılık. Aranan: '110'
  d := Grover('110')
  sayım := d.Çalıştır(200)
  yaz('Sonuçlar: ' + sayım, 1)
  yaz(Biçimle('Doğru cevabın olasılığı: %{:.1}', d.Olasılık('110') * 100), 1)
bitir.
```

```
Sonuçlar: ['000': 1, '001': 1, '010': 3, '011': 2, '100': 2, '101': 1, '110': 186, '111':
    ↪ 4]
Doğru cevabın olasılığı: %94.5
```

Algoritma iki adımı tekrarlar. **Kehanet** aranan durumun genliğinin işaretini çevirir. **Yayılım** tüm genlikleri ortalamaları etrafında yansıtır. Her turda doğru cevabın genliği büyür, ötekiler küçülür. Bu iki adımın Yöntem kodunu `kutuphane/KuantumAlgoritmalari.y` dosyasında okuyabilirsiniz.

Kuantum ve yapay zekâ buluşuyor: değişimsel devreler

Parametreli kapılar (örneğin `Ry(q, θ)`) bir sinir ağının ağırlıkları gibi ayarlanabilir. `ParametreKaydırma`, bir devrenin çıktısının parametrelere göre gradyanını hesaplar. Ardından Bölüm 12'deki gradyan inişi aynen uygulanır.

```
program DeğişimselDevre
kullan Kuantum, Matematik
değişken
  açı: liste
  grad: liste
  i: tamsayı

// Ry(θ) uygulanmış bir kübitin ⟨Z⟩ değeri = cos θ. En küçük değer θ = π'de: -1
fonksiyon Enerji(p: liste): ondalık
değişken
  d: herhangi
başla
  d := Devre(1)
  d.Ry(0, p[0])
  dön d.BeklenenDeğer('Z')
bitir

başla
  açı := [0.3]
  1'den 30'a kadar her i için
  başla
    grad := ParametreKaydırma(@Enerji, açı)
    açı := [açı[0] - 0.4 * grad[0]]
    eğer i mod 10 = 0 ise
      yaz(Biçimle('adım {:2}: θ = {:.4}  enerji = {:.4}', i, açı[0], Enerji(açı)), 1)
  bitir
  yaz(Biçimle('π = {:.4}', pi), 1)
bitir.
```

```
adım 10: θ = 2.9135  enerji = -0.9741
adım 20: θ = 3.1402  enerji = -1.0000
adım 30: θ = 3.1416  enerji = -1.0000
π = 3.1416
```

Devre, enerjiyi en aza indiren açıyı kendi kendine buldu: $\theta = \pi$. Moleküllerin taban durum enerjisini hesaplayan **VQE** ve kuantum makine öğrenmesi modelleri bu fikrin büyütülmüş halleridir. İki kübitli tam bir VQE örneği, kuantum ışınlama ve QFT için `ornekler/kuantum.y` dosyasına bakabilirsiniz. `@Enerji`, fonksiyonu çağırmadan değer olarak verir (Bölüm 8).

Alıştırmalar

1. Üç kübitlik bir GHZ durumu (`0.7071|000⟩ + 0.7071|111⟩`) kuran bir devreyi `H` ve `CX` kapılarıyla kendiniz yazınız. Sonra `GHZ(3)` ile karşılaştırınız.
2. `Grover` fonksiyonunu 4 kübitle (örneğin `'1011'`) çalıştırınız. `tekrar` parametresini 1, 2, 3, 4 yapıp doğru cevabın olasılığının nasıl değiştiğini gözlemleyiniz.

3. `Enerji` fonksiyonunu `d.Ry(0, p[0]); d.Rz(0, p[1]);` ve `BeklenenDeğer('X')` olacak şekilde değiştiriniz ve iki parametreyi birlikte eğitiniz.

EK

A

Anahtar sözcükler

Yöntem	Pascal karşılığı	Kullanımı
`program`, `birim`	program, unit	Dosyanın türü ve adı
`kullan`	uses	Birim yükler
`arayüz`, `uygulama`	interface, implementation	Birimin dış ve iç yüzü
`sabit`, `değişken`, `tip`	const, var, type	Bildirim bölümleri
`prosedür`, `fonksiyon`, `kurucu`	procedure, function, constructor	Alt programlar
`başla … bitir`	begin ... end	Komut bloğu
`eğer … ise … değilse`	if ... then ... else	Karar
`x değerine göre … bitir`	case ... of ... end	Çok seçenekli karar
`koşul iken …`	while ... do	Koşullu döngü
`a'dan b'ye kadar her i için …`	for ... to ... do	Sayan döngü (birer birer)

Yöntem	Pascal karşılığı	Kullanımı
`a'dan b'ye adımda c artan / azalan her i için …`	for … to / downto … do	Adımlı ve geriye sayan döngü
`tekrarla … koşul olana dek`	repeat … until	En az bir kez çalışan döngü
`L içindeki her x için …`	for … in … do	Koleksiyon döngüsü
`n kez tekrarla …`	---	Belirli sayıda tekrar
`kır`, `devam`	break, continue	Döngü denetimi
`dön`, `çık`	Result + Exit, Exit	Alt programdan çıkış
`dene`, `yakala`, `sonunda`, `fırlat`	try, except, finally, raise	Hata yönetimi
`sınıf`, `kayıt`, `miras`, `kendi`	class, record, inherited, Self	Nesneler
`ve`, `veya`, `yada`, `değil`	and, or, xor, not	Mantıksal işleçler
`böl`, `mod`	div, mod	Tam sayı bölmesi ve kalan
`doğru`, `yanlış`, `boş`	True, False, nil	Sabit değerler
`… dizisi`	array of …	`… listesi` ile aynıdır: `tamsayı dizisi`; boyutu sabitlemek için `L.Uzunluk(7, 1)`

Bu sözcükler değişken adı olarak kullanılamaz. `adım`, `kez`, `olana dek`, `değerine göre`, `özel`, `genel`, `sanal`, `ezer` gibi sözcükler ise yalnızca kendi yerlerinde anlam taşır ve ad olarak da kullanılabilir.

EK

B

Sık yapılan hatalar

Belirti	Neden	Çözüm
`';' ya da satır sonu bekleniyordu`	Aynı satıra iki komut bitişik yazılmış ya da bir komut yarım kalmış	Komutlar ayrı satırlara yazılmalı ya da `;` ile ayrılmalıdır
Döngü yalnızca ilk komutu tekrarlıyor	Gövdede birden çok komut var ama `başla … bitir` yok	Gövde `başla … bitir` içine alınmalıdır
`Bitiş değerinde yönelme eki bekleniyordu`	`1'den 10 kadar` gibi ek unutulmuş	`1'den 10'a kadar` yazılmalıdır
`Sayaçlı döngü şöyle yazılır…`	Eski `için i := 1 kadar 10 yap` biçimi	`1'den 10'a kadar her i için`
`bu kapsamda zaten tanımlı`	`x`/`X` ya da `sayaç`/`sayac` gibi aynı sayılan iki ad	Adlardan biri değiştirilmelidir
`... çağrılamaz`	Bir değişken, aynı adlı yerleşik işlevi gölgeliyor (`toplam` → `Toplam`)	Değişkene başka bir ad verilmelidir

Belirti	Neden	Çözüm
`anahtar sözcüktür ve ad olarak kullanılamaz`	`böl`, `tip`, `kayıt` gibi bir sözcük ad yapılmış	Başka bir ad seçilmelidir
`Atama için '=' değil ':=' kullanın`	`x = 5` yazılmış	`x := 5`
`tip uyuşmazlığı`	Tamsayı değişkene ondalık, metne sayı atanmış	`tamsayı(...)`, `metin(...)` ile çevrilmelidir
`Metin dizini sınır dışı: 0`	Metin 0'dan başlatılmış	Metinler 1'den, listeler 0'dan başlar
`Koşul mantıksal olmalı`	`eğer x ise` gibi sayı koşul yapılmış	`eğer x <> 0 ise`
`'Toplam: 23'` gibi tuhaf metin	`'Toplam: ' + 2 + 3` soldan sağa birleşti	Hesap paranteze alınmalıdır: `'Toplam: ' + (2 + 3)`
`Blok sonu bekleniyordu — 'son' yerine 'bitir'`	Eski söz dizimi	Bloklar `bitir` ile kapatılmalıdır
`'yazsatır' tanımsız`	Eski söz dizimi	`yaz(değer, 1)`
`Girdi sonuna ulaşıldı`	`oku` okunacak satır bulamadı	Girdi kontrol edilmelidir

EK

C

Alıştırma çözümleri

Aşağıdaki çözümlerin hepsi çalıştırılmıştır. Bir alıştırmanın birden çok doğru çözümü olabilir; sizinki farklı olsa da doğru sonucu veriyorsa bir sorun yoktur.

Bölüm 3

1. Daire

```
program Daire
değişken
  r: ondalık
başla
  yaz('Yarıçap: ')
  oku(r, 1)
  yaz(Biçimle('Alan  : {:.2}', pi * r * r), 1)
  yaz(Biçimle('Çevre : {:.2}', 2 * pi * r), 1)
bitir.
```

```
Yarıçap: 2.5
Alan  : 19.63
Çevre : 15.71
```

2. Süre çevirme

```
program SüreÇevir
değişken
  toplam, saat, dakika, saniye: tamsayı
başla
  yaz('Saniye: ')
  oku(toplam, 1)
  saat := toplam böl 3600
  dakika := (toplam mod 3600) böl 60
  saniye := toplam mod 60
  yaz(saat + ' saat ' + dakika + ' dakika ' + saniye + ' saniye', 1)
bitir.
```

```
Saniye: 3725
1 saat 2 dakika 5 saniye
```

Bölüm 5

1. FizzBuzz (kısa olsun diye 15'e kadar)

```
program FizzBuzz
değişken
  i: tamsayı
başla
  1'den 15'e kadar her i için
    eğer i mod 15 = 0 ise yaz('FizzBuzz ')
    değilse eğer i mod 3 = 0 ise yaz('Fizz ')
    değilse eğer i mod 5 = 0 ise yaz('Buzz ')
    değilse yaz(i + ' ')
  yaz('', 1)
bitir.
```

```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz
```

15 kontrolü başta olmalıdır; yoksa 15 için önce `Fizz` koşulu tutar.

2. Sıfıra kadar oku

```
program SıfıraKadar
değişken
  n, adet: tamsayı
  toplam: ondalık
başla
  toplam := 0;  adet := 0
  tekrarla
    yaz('Sayı (bitirmek için 0): ')
    oku(n, 1)
    eğer n <> 0 ise
    başla
      toplam + = n
      adet + = 1
    bitir
  n = 0 olana dek
  eğer adet > 0 ise
    yaz('Toplam = ' + toplam + ', ortalama = ' + toplam / adet, 1)
  değilse
    yaz('Hiç sayı girilmedi.', 1)
bitir.
```

```
Sayı (bitirmek için 0): 4
Sayı (bitirmek için 0): 8
Sayı (bitirmek için 0): 6
Sayı (bitirmek için 0): 0
Toplam = 18.0, ortalama = 6.0
```

Bölüm 8

1. Ünlü sayısı

```
program Ünlüler

fonksiyon ÜnlüSayısı(s: metin): tamsayı
değişken
  c: herhangi
başla
  sonuç := 0
  KüçükHarf(s) içindeki her c için
    eğer c içinde 'aeıioöüü' ise sonuç + = 1
bitir

başla
  yaz(ÜnlüSayısı('Yöntem Programlama Dili'), 1)
bitir.
```

```
8
```

`içinde` metinlerde de çalışır: `c içinde 'aeıioöuü'` karakterin ünlüler arasında olup olmadığını sorar.

2. Fibonacci

```
program Fibonacci
değişken
  fib: liste
  i: tamsayı
başla
  fib := [0, 1]
  2'den 14'e kadar her i için
    fib.Ekle(fib[i - 1] + fib[i - 2])
  yaz(fib, 1)
  yaz('Çiftlerin toplamı: ' + (fib |> Süz(fonksiyon(x) => x mod 2 = 0) |> Toplam), 1)
bitir.
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
Çiftlerin toplamı: 188
```

3. Başarılı öğrenciler

```
program Başarılılar
değişken
  notlar: sözlük
  ad: herhangi
başla
  notlar := ['Mert': 65, 'Ayşe': 92, 'Can': 78, 'Defne': 88, 'Bora': 54]
  Sırala(Süz(Anahtarlar(notlar), fonksiyon(a) => notlar[a] > 70)) içindeki her ad
      ↪ için
    yaz(ad + ': ' + notlar[ad], 1)
bitir.
```

```
Ayşe: 92
Can: 78
Defne: 88
```

Bölüm 11

1. Dikdörtgen ve Kare

```
program Şekiller
tip
  Dikdörtgen = sınıf
    en, boy: ondalık
    kurucu Oluştur(e, b: ondalık)
    başla en := e;  boy := b; bitir
    fonksiyon Alan: ondalık;  başla dön en * boy; bitir
    fonksiyon Çevre: ondalık; başla dön 2 * (en + boy); bitir
  bitir

  Kare = sınıf(Dikdörtgen)
    kurucu Oluştur(kenar: ondalık)
    başla miras Oluştur(kenar, kenar); bitir
  bitir

değişken
  k: Kare
başla
  k := Kare.Oluştur(4)
  yaz('Alan = ' + k.Alan + ', çevre = ' + k.Çevre, 1)
bitir.
```

```
Alan = 16.0, çevre = 16.0
```

2. İpucu: Sınıfa `işlemler: liste;` alanı ekleyip kurucuda `işlemler := [];` yapabilirsiniz. `Yatır` ve `Çek` içinde `işlemler.Ekle('Yatır ' + miktar);` gibi bir kayıt düşmeniz yeterlidir. `Geçmiş` metodu ise `dön işlemler;` olabilir. Alanın adı metodun adıyla aynı olmamalı; bu yüzden alana `geçmiş` değil `işlemler` dedik.

3. İpucu: `Şekiller.y` dosyasını `birim Şekiller;` ile başlatabilirsiniz. Sınıflar `arayüzdeki` `tip` bölümüne, metod gövdeleri ise `uygulamaya` `Dikdörtgen.Alan` biçiminde yazılır. Programı aynı klasöre koyup `kullan Şekiller;` demeniz yeterlidir.

Bölüm 14

1. Eğri uydurma

```
program EğriUydurma
kullan YZ
değişken
  xs, ys: liste
  i: tamsayı
  x: ondalık
  model, opt: herhangi
  geçmiş: liste
```

```
başla
  Tohum(0)
  xs := [];  ys := []
  0'dan 19'a kadar her i için
  başla
    x := -1 + i * 2 / 19
    xs.Ekle([x])
    ys.Ekle([2 * x * x - 1])
  bitir
  model := Ardışık([Yoğun(1, 16, 'tanh'), Yoğun(16, 1)])
  opt := Adam(model.Parametreler, 0.02)
  geçmiş := Eğit(model, xs, ys, 'okh', opt, 500)
  yaz(Biçimle('Son kayıp: {:.5}', geçmiş[Uzunluk(geçmiş) - 1]), 1)
  yaz(Biçimle('x = 0.5 için tahmin {:.3} (gerçek -0.5)', model.Tahmin([[0.5]])[0,
      ↪ 0]), 1)
bitir.
```

```
Son kayıp: 0.00030
x = 0.5 için tahmin -0.496 (gerçek -0.5)
```

Sayı tahmininde son katmanda etkinleştirme kullanılmaz; çıktı her değeri alabilmelidir.

2–3. İpucu: Bölüm 14'teki programlardan yola çıkabilirsiniz. Spam filtresinde `filtre.Olasılıklar(m)` bir sözlük döndürür: `['spam': 0.93, 'normal': 0.07]` gibi.

Bölüm 15

1. GHZ durumu

```
program GHZÖrneği
kullan Kuantum, KuantumAlgoritmaları
değişken
  d: herhangi
başla
  d := Devre(3)
  d.H(0)
  d.CX(0, 1)
  d.CX(1, 2)
  yaz(d.Durum, 1)
  yaz('Aynı mı? ' + (Sadakat(d, GHZ(3)) > 0.9999), 1)
bitir.
```

```
0.7071|000⟩ + 0.7071|111⟩
Aynı mı? doğru
```

`Sadakat` iki kuantum durumunun ne kadar benzediğini 0 ile 1 arasında ölçer.

2. Grover'da tekrar sayısı

```
program GroverTekrar
kullan Kuantum, KuantumAlgoritmaları
değişken
  t: tamsayı
başla
  1'den 4'e kadar her t için
    yaz(Biçimle('tekrar = {}: P(1011) = {:.3}', t, Grover('1011',
        ↪ t).Olasılık('1011')), 1)
bitir.
```

```
tekrar = 1: P(1011) = 0.473
tekrar = 2: P(1011) = 0.908
tekrar = 3: P(1011) = 0.961
tekrar = 4: P(1011) = 0.582
```

Olasılık 3 tekrarda en yüksek değerine ulaşır, sonra düşer. Grover algoritmasında fazla tekrar zararlıdır; en iyi sayı yaklaşık $(\pi/4)\cdot\sqrt{N}$'dir. 16 olasılık için bu 3'e denk gelir.

3. İpucu: `açı := [0.3, 0.1];` ile başlayabilirsiniz. `ParametreKaydırma` iki elemanlı bir gradyan listesi döndürür; güncellemeyi Bölüm 8'deki gibi `Eşle(Eşleştir(açı, grad), fonksiyon(p) => p[0] - 0.4 * p[1])` ile yapabilirsiniz.

Tebrikler, kitabı bitirdiniz. Buradan sonra Yöntem klasöründeki `ornekler` ve `kutuphane` klasörlerini okuyabilirsiniz: hepsi bu kitapta öğrendiğiniz yapılarla yazılmıştır. Dilin tam başvuru belgesi `README.md` dosyasındadır.