Yöntem Birimler ve Araçlar Kılavuzu

ÖNDER ÖRNEK

Yöntem Birimler ve Araçlar Kılavuzu

BİRİNCİ BASKI
2026

Yöntem Birimler ve Araçlar Kılavuzu
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

Bu kılavuz ne anlatır? 1

1. Pencere: arayüz 3

2. Mobil: telefon özellikleri 7

3. Güvenlik: saldırılara karşı savunma 11

4. Takvim: tarih ve saat 15

5. Desen: düzenli ifadeler 17

6. Ağ: web servislerine bağlanmak 21

7. Veritabanı: SQL veritabanları 23

8. Sunucu: web servisleri 27

9. Üç katmanlı bir uygulama 31

10. Araçlar 37

11. Birimleri birlikte kullanmak 47

Bu kılavuz ne anlatır?

Yöntem'in çekirdeği küçüktür; gerisi **birimlerle** gelir. Bir birimi kullanmak için programın başına `kullan` yazmak yeterlidir, ayrıca bir şey kurulmaz (tek istisna, PostgreSQL ve MySQL'e bağlanmak için gereken sürücülerdir). Bu kılavuz gündelik programlarda en çok işe yarayan sekiz birimi ve dille birlikte gelen araçları tanıtır. Dilin kendisini öğrenmek için *Yöntem ile Programlamaya Giriş* kitabına, yapay zekâ birimleri için *Yöntem ile Makine Öğrenmesi* ve *Yöntem ile Sıfırdan Dil Modeli* kitaplarına bakabilirsiniz.

Birim	Ne işe yarar	Bölüm
`Pencere`	Düğme, kutu ve tuvallerle arayüz	1
`Mobil`	Telefon özellikleri: titreşim, bildirim, konum, kalıcı depo	2
`Güvenlik`	Parola saklama, şifreleme, girdi denetimi, oturum belirteçleri	3
`Takvim`	Tarih ve saat hesapları	4
`Desen`	Düzenli ifadelerle metin arama ve değiştirme	5
`Ağ`	Web servislerine HTTP istekleri	6
`Veritabanı`	SQLite, PostgreSQL ve MySQL veritabanlarına SQL sorguları	7

Birim	Ne işe yarar	Bölüm
`Sunucu`	HTTP isteklerini karşılayan web servisleri	8

9. bölüm bu ikisiyle üç katmanlı bir uygulama kurar. Araçlar 10. bölümdedir: test çalıştırıcı, kod biçimlendirici, proje oluşturucu, paketleyici ve VS Code eklentisi.

Bu kılavuzdaki her program gerçekten çalıştırılmıştır; altındaki çıktı onun kendi çıktısıdır. Programlar deponun `birim_kilavuzu/` klasöründedir ve iki uygulamada da (JavaScript derleyicisi ve Python yorumlayıcısı) aynı çıktıyı verir. Yalnızca 6. bölümdeki ağ örneği bir internet servisine bağlandığı için çıktısıyla birlikte verilmemiştir.

Bütün birimler için geçerli üç kural

- **Adlar büyük/küçük harfe ve Türkçe karakterlere duyarsızdır.** `Şimdi`, `şimdi` ve `SIMDI` aynı addır. Bu yüzden bir birim üyesiyle aynı adda değişken tanımlamamak gerekir: `şimdi := Şimdi` yazarsanız değişken, `Şimdi` işlevini gölgeler. Değişkene başka bir ad verebilir (`an`) ya da üyeyi birim adıyla çağırabilirsiniz (`Takvim.Şimdi`).
- **Testler için benzetim.** Saate, rastgeleliğe ya da bir cihaza bağlı birimlerde çıktıyı sabitleyen yardımcılar vardır (`BenzetimZamanı`, `Tıkla`, `BenzetimKonum`…). Böylece aynı program hem gerçek ortamda çalışır hem de `yontem test` ile sınanabilir.
- **Gömülü arka uç.** Bu sekiz birim ESP32 için üretilen C++ kodunda yoktur; orada `Donanım` birimi kullanılır.

1. Pencere: arayüz

`Pencere` birimi (İngilizce adıyla `UI`) düğme, giriş kutusu, etiket ve tuvallerden arayüz kurar. Aynı program üç yerde çalışır:

Nerede	Nasıl	Ne olur
Oyun alanı	Çalıştır'a basılır	Arayüz **Uygulama** sekmesinde canlı açılır
Web ya da telefon	`yontem paketle program.y`	Tek başına çalışan bir web uygulaması üretilir
Komut satırı ve testler	`yontem program.y`	Benzetim: olayları programın kendisi üretir, arayüz metin olarak yazılır

Bir arayüz programı üç adımdan oluşur: bileşenler oluşturulur, olaylara prosedür bağlanır ve bileşenler `Yerleştir` ile pencereye konur.

```
program Selamlayıcı
{ Pencere birimi: bir giriş kutusu, bir düğme ve bir etiket.
  Oyun alanında ya da paketlenmiş uygulamada arayüz gerçekten açılır; komut satırında
  ve testlerde olaylar Gir ve Tıkla ile benzetilir. }
kullan Pencere

değişken adKutusu, selamDüğmesi, sonuç: herhangi

prosedür Selamla
başla
  eğer adKutusu.Metin = '' ise sonuç.Metin := 'Önce adını yaz.'
```

```
  değilse sonuç.Metin := 'Merhaba, ' + adKutusu.Metin + '!'
bitir

başla
  Başlık('Selamlayıcı')
  adKutusu := GirişKutusu('Adın')
  selamDüğmesi := Düğme('Selam ver')
  sonuç := Etiket('')
  selamDüğmesi.Tıklanınca(@Selamla)
  Yerleştir(Yatay(adKutusu, selamDüğmesi), sonuç)

  eğer değil Canlı ise
  başla
    Tıkla(selamDüğmesi)
    yaz(sonuç.Metin, 1)
    Gir(adKutusu, 'EdaEce')
    Tıkla(selamDüğmesi)
    yaz(sonuç.Metin, 1)
  bitir
  Çalıştır
bitir.
```

Komut satırındaki çıktısı şöyledir:

```
Önce adını yaz.
Merhaba, EdaEce!
Pencere "Selamlayıcı"
  Yatay
    GirişKutusu "EdaEce" (ipucu "Adın")
    Düğme "Selam ver"
  Etiket "Merhaba, EdaEce!"
```

`Canlı`, program gerçek bir arayüzde çalışıyorsa `doğrudur`. `eğer değil Canlı ise` bloğu yalnızca benzetimde çalışır: `Gir` kutuya metin yazar, `Tıkla` düğmeye basar. Sondaki `Çalıştır` canlı ortamda olayları beklemeye başlar; benzetimde arayüzün son hâlini ağaç olarak yazar. Testler bu ağacı beklenen çıktıyla karşılaştırır.

Bileşenler

Bileşen	Oluşturma	Başlıca üyeleri
Etiket	`Etiket(metin)`	`Metin`
Düğme	`Düğme(metin)`	`Metin`, `Tıklanınca(@P)`
Giriş kutusu	`GirişKutusu(ipucu)`	`Metin`, `İpucu`, `Değişince(@P)`
Onay kutusu	`OnayKutusu(metin, işaretli)`	`İşaretli`, `Değişince(@P)`

Bileşen	Oluşturma	Başlıca üyeleri
Kaydırıcı	`Kaydırıcı(enAz, enÇok, değer)`	`Değer`, `Değişince(@P)`
Seçim kutusu	`SeçimKutusu(seçenekler)`	`Seçili`, `SeçiliMetin`, `Seçenekler`, `Değişince(@P)`
Tuval	`Tuval(genişlik, yükseklik)`	`Renk`, `Çizgi`, `Dikdörtgen`, `Daire`, `Yazı`, `Temizle`, `Dokununca(@P)`
Yerleşim	`Yatay(…)`, `Dikey(…)`	`Ekle`, `Temizle`, `Çocuklar`

Her bileşende `Etkin` ve `Görünür` vardır. Pencere düzeyinde `Başlık(metin)`, `Yerleştir(…)`, `Temizle`, belirli aralıklarla bir prosedürü çağıran `Aralıkla(ms, @P)` ve `AralığıDurdur` bulunur. Benzetim yardımcıları `Tıkla`, `Gir`, `Seç`, `Dokun` ve zamanı ilerleten `İlerle(ms)`'dir.

Olay prosedürleri `@Ad` ile verilir. Kısa işler için adsız prosedür de yazılabilir: `düğme.Tıklanınca(prosedür başla sayı := sayı + 1 bitir)`.

2. Mobil: telefon özellikleri

`Mobil` birimi telefonun titreşim, bildirim, paylaşma, konum, hareket algılayıcısı ve kalıcı depo özelliklerine erişir. Genellikle `Pencere` ile birlikte kullanılır.

```
program SuTakibi
{ Mobil birimi: içilen bardak sayısı telefonda saklanır, hedefe ulaşınca telefon
    ↪ titrer
  ve bildirim gösterir. Benzetimde titreşim ve bildirim çıktıya birer satır olarak
      ↪ yazılır. }
kullan Pencere, Mobil

sabit HEDEF: tamsayı = 3

değişken
  bardak: tamsayı
  gösterge, içDüğmesi: herhangi

prosedür Göster
başla
  gösterge.Metin := Metin(bardak) + ' / ' + Metin(HEDEF) + ' bardak'
bitir

prosedür İç
başla
  bardak := bardak + 1
  Sakla('bardak', bardak)          { uygulama kapanıp açılsa da kalır }
  Göster
  eğer bardak = HEDEF ise
  başla
    Titreşim(300)
```

```
    Bildirim('Tebrikler', 'Bugünkü su hedefine ulaştın')
  bitir
bitir

başla
  Başlık('Su Takibi')
  bardak := Getir('bardak', 0)     { daha önce saklanmamışsa 0 }
  gösterge := Etiket('')
  içDüğmesi := Düğme('Bir bardak içtim')
  içDüğmesi.Tıklanınca(@İç)
  Yerleştir(gösterge, içDüğmesi)
  Göster

  eğer değil Canlı ise
  başla
    yaz('Platform: ' + Platform + ', ekran ' + EkranGenişliği + '×' +
        ↪ EkranYüksekliği, 1)
    3 kez tekrarla Tıkla(içDüğmesi)
    yaz(gösterge.Metin + ' (saklanan: ' + Getir('bardak', 0) + ')', 1)
  bitir
  Çalıştır
bitir.
```

```
Platform: benzetim, ekran 390×844
[Titreşim 300 ms]
[Bildirim] Tebrikler: Bugünkü su hedefine ulaştın
3 / 3 bardak (saklanan: 3)
Pencere "Su Takibi"
  Etiket "3 / 3 bardak"
  Düğme "Bir bardak içtim"
```

Benzetimde `Titreşim`, `Bildirim` ve `Paylaş` çıktıya köşeli ayraçlı birer satır yazar; telefonda gerçekten titreşir ve bildirim gösterir. `Sakla` ile yazılan değerler uygulama kapansa da kalır; metin, sayı, mantıksal değer ve bunların listeleri saklanabilir.

Üye	Ne yapar
`Titreşim(ms)`	Telefonu titretir
`Bildirim(başlık, metin)`	Bildirim gösterir
`Paylaş(metin)`	Paylaşma menüsünü açar
`KonumAl(@P)`	Konum bulununca `P(enlem, boylam)` çağrılır
`HareketOlunca(@P)`	Telefon hareket ettikçe `P(x, y, z)` çağrılır
`YönDeğişince(@P)`	Ekran döndüğünde `P(yön)` çağrılır (`'dikey'` ya da `'yatay'`)

Üye	Ne yapar
`Sakla(anahtar, değer)`, `Getir(anahtar, varsayılan)`, `Unut(anahtar)`	Kalıcı depo
`Platform`, `Dokunmatik`, `EkranGenişliği`, `EkranYüksekliği`, `EkranYönü`	Cihaz bilgisi
`BenzetimKonum`, `BenzetimHareket`, `Salla`, `BenzetimYön`	Testler için benzetim

Telefona kurmak. `yontem paketle program.y --mobil --ad "Su Takibi"` komutu bir klasör üretir: `index.html`, uygulama bildirimi (`manifest.webmanifest`), çevrim dışı çalışma için `sw.js` ve simgeler. Klasörü HTTPS ile yayımlamanız yeterlidir (GitHub Pages, Netlify ya da kendi sunucunuz); telefonun tarayıcısında açıp **Ana ekrana ekle** deyince uygulama olarak kurulur ve internet olmadan da çalışır. Uygulama mağazası paketi (Android/iOS) bu sürümde yoktur.

3. Güvenlik: saldırılara karşı savunma

`Güvenlik` birimi yaygın saldırılara karşı denenmiş yapı taşları sunar. Kendi şifreleme yönteminizi yazmak yerine bunları doğru yerde kullanmanız yeterlidir.

```
program ÜyelikÖzeti
{ Güvenlik birimi: parola saklama, kullanıcı girdisini temizleme ve süreli oturum
    ↪ belirteci.
  Çıktının her çalıştırmada aynı olması için birimin saati sabitlendi. }
kullan Güvenlik

sabit ANAHTAR: metin = 'bunu-ortam-degiskeninden-oku'

değişken kayıtlı, oturum: metin

başla
  BenzetimZamanı(1800000000)
  { 1) Parola düz metin saklanmaz; tuzlu ve yavaşlatılmış özeti saklanır. }
  kayıtlı := ParolaÖzeti('Deniz-Feneri-1907', 1000)   { gerçek sistemde ikinci
      ↪ argümanı vermeyin }
  yaz('Kayıt biçimi : ' + Ayır(kayıtlı, '$')[0] + ', ' + Uzunluk(kayıtlı) + '
      ↪ karakter', 1)
  yaz('Doğru parola : ' + Metin(ParolaDoğrula('Deniz-Feneri-1907', kayıtlı)), 1)
  yaz('Yanlış parola: ' + Metin(ParolaDoğrula('deniz-feneri', kayıtlı)), 1)
  yaz('Zayıf parola : ' + Metin(ParolaSorunları('kedi123')), 1)

  { 2) Kullanıcının yazdığı metin sayfaya ve sorguya kaçışlanarak girer. }
  yaz('HTML : ' + HtmlKaçış('<script>çal()</script>'), 1)
  yaz('SQL  : ' + SqlMetni('x'' OR ''1''=''1'), 1)
  yaz('Dosya: ' + DosyaAdıTemizle('../../etc/passwd') + ', yol güvenli mi: ' +
      ↪ Metin(YolGüvenli('../../etc/passwd')), 1)
```

```
  { 3) Oturum: imzalı ve 30 dakika geçerli belirteç. }
  oturum := BelirteçÜret(ANAHTAR, 'edaece', 1800)
  yaz('Oturum sahibi: ' + Metin(BelirteçDoğrula(ANAHTAR, oturum)), 1)
  yaz('Sahte anahtar: ' + Metin(BelirteçDoğrula('tahmin', oturum)), 1)
  BenzetimZamanı(1800000000 + 1801)
  yaz('31 dk sonra  : ' + Metin(BelirteçDoğrula(ANAHTAR, oturum)), 1)
bitir.
```

```
Kayıt biçimi : pbkdf2-sha256, 116 karakter
Doğru parola : doğru
Yanlış parola: yanlış
Zayıf parola : ['en az 8 karakter olmalı', 'büyük harf yok', 'simge yok']
HTML : &lt;script&gt;çal()&lt;/script&gt;
SQL  : 'x'' OR ''1''=''1'
Dosya: passwd, yol güvenli mi: yanlış
Oturum sahibi: edaece
Sahte anahtar: boş
31 dk sonra  : boş
```

Saldırı	Araç
Parola veritabanının çalınması	`ParolaÖzeti`, `ParolaDoğrula`, `YenidenÖzetlenmeli`, `ParolaGücü`, `ParolaSorunları`
Verinin değiştirilmesi	`Özet`, `İmza`, `İmzaDoğru`, `BelirteçÜret`, `BelirteçDoğrula`
Yanıt süresinden bilgi sızması	`GüvenliEşit`
Verinin okunması	`AnahtarÜret`, `ParoladanAnahtar`, `Şifrele`, `ŞifreÇöz`
Tahmin edilebilir değerler	`Belirteç`, `Tuz`, `Kimlik`, `GüvenliRastgele`
XSS, SQL enjeksiyonu, yol geçişi	`HtmlKaçış`, `SqlMetni`, `YolGüvenli`, `DosyaAdıTemizle`, `UrlGüvenli`, `EpostaGeçerli`, `İzinliKarakterler`
Kaba kuvvet	`Sınırlayıcı(enÇok, saniye)`
İz bırakmadan saldırı	`Günlük`, `GünlükKayıtları`, `Maskele`

Üç temel alışkanlık çoğu açığı kapatır: parolaları `ParolaÖzeti` ile saklamak, kullanıcıdan gelen her metni gittiği yere göre kaçışlamak (`HtmlKaçış`, `SqlMetni`) ve gizli değerleri = ile değil `GüvenliEşit` ile karşılaştırmak.

Birim açık anahtarlı şifreleme ve TLS içermez; ağ trafiğini HTTPS korur. Algoritmalar standart test vektörleriyle doğrulanmıştır ama bağımsız bir güvenlik denetiminden geçmemiştir. Her aracın ayrıntılı anlatımı, tam bir güvenli giriş örneği

ve iyi uygulamalar *Yöntem Güvenlik Birimi — Kullanım Kılavuzu*'ndadır; örnekler `guvenlik_ornekleri/` klasöründedir.

4. Takvim: tarih ve saat

`Takvim` biriminde zaman, 1 Ocak 1970'ten beri geçen saniye sayısıdır (Unix saniyesi) ve bir tamsayıyla tutulur. Bu sayede iki tarih `<` ile karşılaştırılır, sıralanır, birbirinden çıkarılır. Hesaplar UTC'de yapılır; yerel saati yazmak için `YerelFark` (dakika) eklenir.

```
program SınavTakvimi
{ Takvim birimi: tarih okuma, yazma ve gün hesabı. }
kullan Takvim

değişken
  bugün, sınav, teslim: tamsayı

başla
  BenzetimZamanı(TarihYap(2026, 11, 10, 9, 0, 0), 180)   { testler için sabit saat;
      ↪ UTC+3 }
  bugün := Şimdi
  sınav := TarihOku('15.01.2027', 'GG.AA.YYYY')
  yaz('Bugün : ' + TarihYaz(bugün + YerelFark * 60, 'G AAAA YYYY GGGG, SS:DD'), 1)
  yaz('Sınav : ' + TarihYaz(sınav, 'G AAAA YYYY GGGG') + ' — ' + GünFarkı(bugün,
      ↪ sınav) + ' gün kaldı', 1)
  teslim := AyEkle(GünBaşı(bugün), 1)
  yaz('Ödev teslimi (bir ay sonra): ' + TarihYaz(teslim, 'GG.AA.YYYY') + ', ' +
      ↪ HaftaNo(teslim) + '. hafta', 1)
  eğer HaftaGünü(teslim) > 5 ise yaz('  Hafta sonuna denk geliyor; pazartesi: ' +
      ↪ TarihYaz(GünEkle(teslim, 8 - HaftaGünü(teslim)), 'GG.AA.YYYY'), 1)
  yaz('Şubat 2028 ' + AydakiGünSayısı(2028, 2) + ' gün; 2028 artık yıl mı: ' +
      ↪ Metin(ArtıkYıl(2028)), 1)
bitir.
```

```
Bugün : 10 Kasım 2026 Salı, 12:00
Sınav : 15 Ocak 2027 Cuma — 66 gün kaldı
Ödev teslimi (bir ay sonra): 10.12.2026, 50. hafta
Şubat 2028 29 gün; 2028 artık yıl mı: doğru
```

Kalıplar. `TarihYaz` ve `TarihOku` aynı simgeleri kullanır:

Simge	Anlamı	Örnek
`YYYY`, `YY`	Yıl	2026, 26
`AA`, `A`	Ay (iki haneli, yalın)	03, 3
`AAAA`, `AAA`	Ay adı, kısa ay adı	Mart, Mar
`GG`, `G`	Gün (iki haneli, yalın)	05, 5
`GGGG`, `GGG`	Gün adı, kısa gün adı	Perşembe, Per
`SS`, `DD`, `SN`	Saat, dakika, saniye	09, 05, 30

Küçük harfler ve noktalama olduğu gibi yazılır. Büyük harfli düz metin köşeli ayraca alınır: `'[Saat] SS:DD'`. Kalıp verilmezse `TarihOku` `2026-11-10`, `2026-11-10 19:30` ve `2026-11-10 19:30:00` biçimlerini tanır.

Üye	Ne yapar
`Şimdi`, `YerelFark`	Şu an (Unix saniyesi) ve yerel saatin UTC'den farkı (dakika)
`TarihYap(yıl, ay, gün [, saat, dakika, saniye])`	Tarihten zaman üretir; geçersiz tarihte hata verir
`Parçala(zaman)`	`yıl`, `ay`, `gün`, `saat`, `dakika`, `saniye`, `haftaGünü`, `yılGünü` içeren sözlük
`TarihYaz(zaman [, kalıp])`, `TarihOku(metin [, kalıp])`	Metne çevirir, metinden okur
`SaniyeEkle`, `GünEkle`, `AyEkle`, `YılEkle`	Zamana ekler (eksi sayı çıkarır); 31 Ocak + 1 ay = şubatın son günü
`GünFarkı(başlangıç, bitiş)`, `GünBaşı(zaman)`	Takvim günü farkı; günün 00:00'ı
`HaftaGünü` (1 = Pazartesi), `HaftaNo` (ISO haftası), `GünAdı`, `AyAdı`	Takvim bilgisi
`ArtıkYıl(yıl)`, `AydakiGünSayısı(yıl, ay)`, `GeçerliTarih(yıl, ay, gün)`	Denetimler
`BenzetimZamanı(zaman [, fark])`	Testler için saati sabitler

Birim 1–9999 yılları arasında çalışır. Saat dilimi adları ve yaz saati kuralları yoktur; yalnızca UTC ve dakika cinsinden sabit bir fark vardır.

5. Desen: düzenli ifadeler

Düzenli ifade (regex), aranan metnin biçimini tarif eden küçük bir dildir: “iki rakam, iki nokta, iki rakam” gibi. `Desen` birimi bunlarla doğrulama, arama, ayıklama ve değiştirme yapar.

```
program SiparişAyıklama
{ Desen birimi: düzenli ifadelerle doğrulama, ayıklama ve değiştirme. }
kullan Desen

sabit YAZI: metin = 'Sipariş #A-1042: 3 adet çay, 12 adet simit; iletişim:
    ↪  edaece@ornek.de, 0532 123 45 67'

değişken g: herhangi

başla
  yaz('Sipariş no : ' + Bul(YAZI, '#[A-Z]-\d+'), 1)
  yaz('Adetler    : ' + Metin(BulHepsi(YAZI, '\d+ adet')), 1)
  TümGruplar(YAZI, '(\d+) adet (\w+)') içindeki her g için
    yaz('  ' + g[2] + ' × ' + g[1], 1)
  yaz('E-posta    : ' + Bul(YAZI, '[\w.]+@[\w.]+\.\w+'), 1)
  yaz('Telefon mu : ' + Metin(TamEşleşir('0532 123 45 67', '0\d{3} \d{3} \d{2}
      ↪  \d{2}')), 1)
  yaz('Gizlenmiş  : ' + DesenDeğiştir(YAZI, '(\d{4}) \d{3} \d{2} (\d{2})', '$1 *** **
      ↪  $2'), 1)
  yaz('Parçalar   : ' + Metin(DesenAyır('çay; simit ,peynir', '\s*[;,]\s*')), 1)
  yaz('Harf ayrımı: ' + Metin(Eşleşir(YAZI, 'SİPARİŞ')) + ', "i" ile: ' +
      ↪  Metin(Eşleşir(YAZI, 'SİPARİŞ', 'i')), 1)
bitir.
```

```
Sipariş no : #A-1042
Adetler    : ['3 adet', '12 adet']
  çay × 3
  simit × 12
E-posta    : edaece@ornek.de
Telefon mu : doğru
Gizlenmiş  : Sipariş #A-1042: 3 adet çay, 12 adet simit; iletişim: edaece@ornek.de, 0532
    ↪ *** ** 67
Parçalar   : ['çay', 'simit', 'peynir']
Harf ayrımı: yanlış, "i" ile: doğru
```

Desenleri tek tırnak içinde yazmanızı öneririz; tek tırnaklı metinlerde \ olduğu gibi kalır.

Desen	Anlamı
`a`, `çay`	Kendisi
`.`	Satır sonu dışında herhangi bir karakter
`\d`, `\w`, `\s`	Rakam; harf, rakam ya da _; boşluk. Büyük harflisi tersidir (`\D`)
`[abc]`, `[a-z]`, `[^0-9]`	Kümedeki bir karakter; aralık; kümede olmayan
`x*`, `x+`, `x?`	Sıfır ya da çok; bir ya da çok; sıfır ya da bir
`x{3}`, `x{2,4}`, `x{2,}`	Tam 3; 2 ile 4 arası; en az 2
`x*?`, `x+?`	Tembel: olabildiğince az
`(…)`, `(?:…)`	Yakalayan grup; yakalamayan grup
`a|b`	a ya da b
`^`, `$`, `\b`	Metnin başı; sonu; sözcük sınırı
`\.`, `\(`, `\\`	Özel karakterin kendisi

`\w` Türkçe harfleri de kapsar. Son argüman olarak `'i'` verilirse büyük/küçük harf ayrımı kalkar ve Türkçedeki dört i (`İ I ı i`) birbirine eş sayılır.

Üye	Ne döndürür
`Eşleşir(metin, desen)`	Desen metnin bir yerinde geçiyor mu
`TamEşleşir(metin, desen)`	Metnin tamamı desene uyuyor mu (doğrulama için)
`Bul(metin, desen)`	İlk eşleşme ya da `boş`
`BulHepsi(metin, desen)`	Tüm eşleşmelerin listesi
`Gruplar(metin, desen)`	İlk eşleşme için `[tamamı, 1. grup, 2. grup…]` ya da `boş`
`TümGruplar(metin, desen)`	Her eşleşme için böyle bir liste

Üye	Ne döndürür
`DesenKonumu(metin, desen)`	İlk eşleşmenin başladığı karakter (yoksa 0)
`DesenDeğiştir(metin, desen, yeni)`	Tüm eşleşmeleri değiştirir; `$1` grup, `$0` eşleşmenin tamamı, `$$` dolar işareti
`DesenAyır(metin, desen)`	Metni eşleşmelerden böler
`DesenKaçış(metin)`, `DesenGeçerli(desen)`	Özel karakterleri etkisizleştirir; deseni denetler

Birim kendi eşleştirme motorunu kullanır, bu yüzden her ortamda aynı sonucu verir. İleri/geri bakış (`(?=…)`) ve geri başvuru (`\1`) yoktur. İç içe tekrar içeren desenler (`(a+)+`) bazı metinlerde çok uzun sürebilir; birim böyle durumda programı kilitlemek yerine hatayla durdurur. Kullanıcının yazdığı metni desenin içine koyacaksanız önce `DesenKaçış` ile kaçışlamanız gerekir.

6. Ağ: web servislerine bağlanmak

`Ağ` birimi HTTP istekleri gönderir. İstek eşzamanlıdır: program yanıt gelene kadar bekler. Yanıt bir sözlüktür: `durum` (200, 404…), `gövde` (metin) ve `başlıklar` (adları küçük harfli bir sözlük).

```
program HavaDurumu
kullan Ağ, Veri

değişken
  yanıt: sözlük
  veri: herhangi

başla
  dene
    yanıt := HttpAl('https://api.ornek.de/hava?' + SorguYap(['şehir': 'İzmir',
        ↪ 'birim': 'C']))
    eğer yanıt['durum'] = 200 ise
    başla
      veri := JSONÇöz(yanıt['gövde'])
      yaz('İzmir: ' + veri['sıcaklık'] + ' °C', 1)
    bitir
    değilse yaz('Servis ' + yanıt['durum'] + ' döndürdü', 1)
  yakala (h)
    yaz('Bağlanılamadı: ' + h.mesaj, 1)
  bitir
bitir.
```

Bu program örnek bir adrese bağlandığı için çıktısı verilmemiştir; adresi kullanacağınız servisinkiyle değiştirebilirsiniz. Dikkat edilecek iki nokta vardır: 404 ya

da 500 gibi yanıtlar hata **değildir**, `durum` ile denetlenir; sunucuya hiç ulaşılamazsa ya da süre dolarsa `dene … yakala` ile yakalanabilen bir çalışma hatası oluşur.

Üye	Ne yapar
`HttpAl(adres [, başlıklar, zamanAşımı])`	GET isteği
`HttpGönder(adres, gövde [, başlıklar, zamanAşımı])`	POST isteği
`HttpİstekYap(yöntem, adres [, gövde, başlıklar, zamanAşımı])`	GET, POST, PUT, PATCH, DELETE ya da HEAD
`SorguYap(sözlük)`	`a =1&b =x%20y` biçiminde sorgu dizgisi
`UrlKodla(metin)`, `UrlÇöz(metin)`	Adreste güvenle kullanılacak biçime çevirir, geri çözer

JSON göndermek için gövdeyi `Veri.JSONYap` ile üretip türünü başlıkta bildirebilirsiniz:

```
yanıt := HttpGönder('https://api.ornek.de/notlar', JSONYap(['metin': 'süt al']),
                    ['Content-Type': 'application/json', 'Authorization': 'Bearer ' +
                        ↪ anahtar])
```

Zaman aşımı saniyedir (varsayılan 30). Yanıt en çok 10 MB olabilir ve UTF-8 metin olarak okunur; dosya indirme, akış ve WebSocket yoktur. Başlık değerlerinde yalnızca ASCII karakterler kullanılabilir; Türkçe karakterleri `UrlKodla` ile kodlamanız gerekir. API anahtarlarını programa yazmak yerine `Sistem.Ortam('ANAHTAR')` ile ortam değişkeninden okumanızı öneririz.

Komut satırında ve Python'da her adrese bağlanılır. Tarayıcıda (oyun alanı, paketlenmiş uygulamalar) yalnızca buna izin veren sitelere (CORS) bağlanılır ve zaman aşımı uygulanmaz.

7. Veritabanı: SQL veritabanları

`Veritabanı` birimi SQL veritabanlarına tek bir arayüzle bağlanır. Hangi veritabanına bağlanılacağını `Bağlan`'a verilen adres belirler; programın geri kalanı aynı kalır.

Adres	Veritabanı	Gerekli sürücü
`sqlite:magaza.db`	SQLite dosyası (yoksa oluşur)	Yok; Node.js 22.5+ ve Python ile gelir
`sqlite::bellek:`	Bellekte geçici SQLite	Yok
`postgres://…`	PostgreSQL	`npm install pg` · `pip install pg8000`
`mysql://…`	MySQL ve MariaDB	`npm install mysql2` · `pip install pymysql`

Sunucu adresinin tamamı `postgres://kullanıcı:parola@sunucu:5432/veritabanı` biçimindedir; parolada özel karakter varsa `Ağ.UrlKodla` ile kodlanır. Sürücü JavaScript derleyicisi için projenin klasörüne `npm` ile, Python yorumlayıcısı için `pip` ile kurulur.

```
program VeritabanıÖrneği
{ Veritabanı birimi: tablo oluşturma, parametreli sorgular ve işlem (transaction).
  'sqlite::bellek:' geçici bir veritabanıdır; kalıcı dosya için 'sqlite:magaza.db'
      ↪ yazın. }
kullan Veritabanı

değişken
  vt: herhangi
```

```
  satır: sözlük

başla
  vt := Bağlan('sqlite::bellek:')
  vt.Çalıştır('CREATE TABLE urun (no INTEGER PRIMARY KEY, ad TEXT NOT NULL UNIQUE,
      ↪ fiyat REAL, adet INTEGER)')
  vt.Çalıştır('INSERT INTO urun (ad, fiyat, adet) VALUES (?, ?, ?)', ['defter', 12.5,
      ↪ 40])
  vt.Çalıştır('INSERT INTO urun (ad, fiyat, adet) VALUES (?, ?, ?)', ['kalem', 4,
      ↪ 120])
  yaz('Son eklenen ürünün numarası: ' + vt.SonKimlik, 1)

  { Değerler SQL metnine eklenmez, '?' ile verilir: kötü niyetli girdi yalnızca bir
      ↪ metin olarak kalır }
  vt.Çalıştır('INSERT INTO urun (ad, fiyat, adet) VALUES (?, ?, ?)', ['x''); DROP
      ↪ TABLE urun; --', 1, 1])
  yaz('Ürün sayısı: ' + vt.TekDeğer('SELECT COUNT(*) FROM urun'), 1)

  vt.Sorgula('SELECT ad, fiyat * adet AS tutar FROM urun WHERE fiyat > ? ORDER BY
      ↪ ad', [3]) içindeki her satır için
    yaz(satır['ad'] + ': ' + satır['tutar'], 1)

  { İşlem: iki değişiklik ya birlikte kalıcı olur ya da hiçbiri }
  vt.İşlemBaşlat
  vt.Çalıştır('UPDATE urun SET adet = adet - ? WHERE ad = ?', [5, 'defter'])
  vt.Çalıştır('UPDATE urun SET adet = adet + ? WHERE ad = ?', [5, 'kalem'])
  vt.Onayla
  satır := vt.TekSatır('SELECT ad, adet FROM urun WHERE ad = ?', ['defter'])
  yaz(satır, 1)

  dene
    vt.Çalıştır('INSERT INTO urun (ad) VALUES (?)', ['defter'])
  yakala (h)
    yaz(h.mesaj, 1)
  bitir
  vt.Kapat
bitir.
```

```
Son eklenen ürünün numarası: 2
Ürün sayısı: 3
defter: 500.0
kalem: 480.0
['ad': 'defter', 'adet': 35]
Veritabanı hatası: UNIQUE constraint failed: urun.ad
```

`Bağlan` bir bağlantı nesnesi döndürür:

Üye	Ne yapar
`vt.Çalıştır(sql [, parametreler])`	Değiştiren komutları çalıştırır; etkilenen satır sayısını döndürür
`vt.Sorgula(sql [, parametreler])`	Satırları sözlük listesi olarak döndürür; anahtarlar sütun adlarıdır
`vt.TekSatır(sql [, parametreler])`	İlk satırı sözlük olarak döndürür; satır yoksa `boş`
`vt.TekDeğer(sql [, parametreler])`	İlk satırın ilk sütununu döndürür; satır yoksa `boş`
`vt.SonKimlik`	Son eklenen satırın otomatik numarası (PostgreSQL'de bunun yerine `INSERT … RETURNING` kullanılır)
`vt.İşlemBaşlat` · `vt.Onayla` · `vt.GeriAl`	İşlem (transaction)
`vt.Kapat` · `vt.Açık` · `vt.Tür`	Bağlantıyı kapatır · açık mı · `'sqlite'`, `'postgres'` ya da `'mysql'`

Değerleri her zaman parametreyle vermeniz gerekir. Sorgudaki `?` işaretlerinin yerine `parametreler` listesindeki değerler sırayla konur; değer SQL metninin parçası olmaz, bu yüzden kullanıcıdan gelen girdi sorguyu değiştiremez (SQL enjeksiyonu). Yer tutucu üç veritabanında da `?` işaretidir. Tablo ve sütun adları parametre olamaz; bir adı değişkenden almak zorundaysanız `SqlAdı(ad)` onu denetleyip tırnaklar.

Tipler. Tamsayı sütunları `tamsayı`, kesirli ve `NUMERIC` sütunları `ondalık`, metin sütunları `metin`, `NULL` ise `boş` olarak gelir. PostgreSQL'in `BOOLEAN` tipi `mantıksal` olur; SQLite ve MySQL'de mantıksal değerler 0 ve 1'dir. Tarih ve saat sütunları veritabanının yazdığı metin olarak gelir (`'2026-11-11 10:30:00'`); `Takvim.TarihOku` ile zamana çevrilebilir.

SQL'in kendisi veritabanından veritabanına biraz değişir (otomatik numara SQLite'ta `INTEGER PRIMARY KEY`, PostgreSQL'de `SERIAL`, MySQL'de `AUTO_INCREMENT` ile bildirilir); birim sorguları çevirmez, yalnızca bağlantıyı, yer tutucuları ve tipleri ortaklaştırır. Veritabanı hataları `Veritabanı hatası: …` iletisiyle fırlatılır ve `dene … yakala` ile yakalanabilir. Parolayı programın içine yazmak yerine `Sistem.Ortam('VT_ADRESI')` ile ortam değişkeninden okumanızı öneririz.

8. Sunucu: web servisleri

`Sunucu` birimi HTTP isteklerini karşılar: bir adrese gelen isteği sizin yazdığınız bir fonksiyona verir, fonksiyonun döndürdüğünü yanıt olarak gönderir.

```
program SunucuÖrneği
{ Sunucu birimi: yollar, yol parametreleri, JSON yanıtı.
  İstekİşle isteği ağ olmadan işler; gerçek sunucu için sondaki Dinle satırını açın.
      ↪ }
kullan Sunucu

değişken
  y: sözlük

fonksiyon AnaSayfa(istek: sözlük): herhangi
başla
  dön '<h1>Merhaba, ' + istek['sorgu'].Al('ad', 'dünya') + '!</h1>'
bitir

fonksiyon Kare(istek: sözlük): herhangi
değişken n: tamsayı
başla
  n := Tamsayı(istek['parametreler']['n'])
  dön JsonYanıt(['sayı': n, 'kare': n * n])
bitir

başla
  Yol('GET', '/', AnaSayfa)
  Yol('GET', '/kare/:n', Kare)

  y := İstekİşle('GET', '/?ad =Ay%C5%9Fe')
```

```
  yaz(Metin(y['durum']) + ' ' + y['gövde'], 1)
  y := İstekİşle('GET', '/kare/12')
  yaz(Metin(y['durum']) + ' ' + y['başlıklar']['content-type'] + ' ' + y['gövde'], 1)
  y := İstekİşle('GET', '/kare/on')
  yaz(Metin(y['durum']) + ' ' + y['gövde'], 1)
  y := İstekİşle('POST', '/kare/3')
  yaz(Metin(y['durum']) + ' ' + y['gövde'], 1)
  y := İstekİşle('GET', '/yok')
  yaz(Metin(y['durum']) + ' ' + y['gövde'], 1)

  { Dinle(8080)      tarayıcıda http://127.0.0.1:8080/kare/12 }
bitir.
```

```
200 <h1>Merhaba, Ayşe!</h1>
200 application/json; charset =utf-8 {"sayı": 12, "kare": 144}
500 Sunucu hatası
405 Bu yöntem desteklenmiyor
404 Bulunamadı
```

İşlev	Ne yapar
`Yol(yöntem, yol, işleyici)`	`'GET'`, `'POST'`, `'PUT'`, `'PATCH'`, `'DELETE'` ya da `'*'` (hepsi) için bir yol tanımlar. `:ad` ile başlayan parça parametredir
`Statik(önEk, klasör)`	Klasördeki dosyaları (HTML, CSS, resim) sunar; klasörün dışına çıkılamaz
`Dinle(port [, adres])`	İstekleri dinlemeye başlar ve program durdurulana kadar bekler
`Yanıt(durum, gövde [, tür])`	Durum kodu ve düz metinle yanıt; `tür` içerik tipidir
`JsonYanıt(değer [, durum])`	Sözlük ya da listeyi JSON olarak gönderir
`Yönlendir(adres [, durum])`	Başka bir adrese yönlendirir (302)
`İstekİşle(yöntem, yol [, gövde, başlıklar])`	İsteği ağ olmadan işler ve yanıtı döndürür; test için
`SunucuyuDurdur`	Bir işleyicinin içinden çağrılır; yanıt gittikten sonra `Dinle` sona erer
`GövdeSınırı(bayt)`	Kabul edilen en büyük istek gövdesi (başlangıçta 1 MB)

İşleyici isteği bir sözlük olarak alır: `istek['yöntem']`, `istek['yol']`, `istek['sorgu']` (adresteki `?ad =değer` çiftleri), `istek['parametreler']` (yoldaki `:ad`

parçaları), `istek['başlıklar']` (adları küçük harfle) ve `istek['gövde']` (metin; JSON ise `Veri.JSONÇöz` ile çözülür). Metin döndürürse bu bir HTML sayfası sayılır; başka her şey için `Yanıt`, `JsonYanıt` ya da `Yönlendir` kullanılır.

Sunucunun kendiliğinden yaptıkları şunlardır:

- Tanımlı olmayan yola `404`, yolu olup yöntemi olmayan isteğe `405`, bozuk adrese `400`, sınırdan büyük gövdeye `413` döner.
- İşleyicide yakalanmayan bir hata olursa istemciye yalnızca `500 Sunucu hatası` gider; hatanın ayrıntısı sunucunun kendi çıktısına yazılır, dışarı sızmaz.
- `Dinle` adres verilmezse yalnızca bu bilgisayardan gelen istekleri kabul eder (`127.0.0.1`). Başka cihazlardan erişim için `Dinle(8080, '0.0.0.0')` yazabilirsiniz.
- İstekler sırayla işlenir: bir işleyici bitmeden sonraki başlamaz. Bu, işleyicilerin birbirinin verisini bozamayacağı anlamına gelir; ama uzun süren bir işleyici diğer istekleri bekletir.

Sunucu doğrudan internete açılacak biçimde tasarlanmamıştır: HTTPS, sıkıştırma ve aynı anda çok sayıda bağlantı için önüne nginx ya da Caddy gibi bir ters vekil koymanız gerekir. Oturum belirteçleri, parola özeti ve istek sınırlama için `Güvenlik` birimi (3. bölüm) hazırdır.

9. Üç katmanlı bir uygulama

`Sunucu` ve `Veritabanı` ile klasik üç katmanlı yapı Yöntem'in birimleriyle doğrudan kurulur: her katman ayrı bir dosyadır ve yalnızca bir altındakini `kullan` ile görür.

Katman	Dosya	Bildiği
Sunum	`10_uc_katman.y` (program)	HTTP: yollar, JSON, durum kodları
İş	`UrunIsleri.y` (birim)	Kurallar: ad, fiyat ve stok denetimi
Depolama	`UrunDeposu.y` (birim)	SQL: tablo ve sorgular

Depolama katmanı yalnızca SQL içerir. `arayüz` bölümü üst katmana ne sunduğunu söyler; `vt` değişkeni `uygulama` bölümünde olduğu için dışarıdan görünmez:

```
birim ÜrünDeposu
{ Depolama katmanı: yalnızca SQL bilir. Kural da ekran da içermez.
  Başka bir veritabanına geçmek için Aç'a verilen adresi değiştirmek yeter. }
kullan Veritabanı

arayüz
  prosedür Aç(adres: metin)
  fonksiyon Hepsi: liste
  fonksiyon Bul(no: tamsayı): herhangi
  fonksiyon AdVarMı(ad: metin): mantıksal
  fonksiyon Ekle(ad: metin; fiyat: ondalık; adet: tamsayı): tamsayı
  prosedür AdetDeğiştir(no, fark: tamsayı)
```

```
uygulama

değişken
  vt: herhangi

prosedür Aç(adres: metin)
başla
  vt := Bağlan(adres)
  vt.Çalıştır('CREATE TABLE IF NOT EXISTS urun (no INTEGER PRIMARY KEY, ad TEXT NOT
      ↪  NULL UNIQUE, fiyat REAL NOT NULL, adet INTEGER NOT NULL)')
bitir

fonksiyon Hepsi: liste
başla
  dön vt.Sorgula('SELECT no, ad, fiyat, adet FROM urun ORDER BY no')
bitir

fonksiyon Bul(no: tamsayı): herhangi
başla
  dön vt.TekSatır('SELECT no, ad, fiyat, adet FROM urun WHERE no = ?', [no])
bitir

fonksiyon AdVarMı(ad: metin): mantıksal
başla
  dön vt.TekDeğer('SELECT COUNT(*) FROM urun WHERE ad = ?', [ad]) > 0
bitir

fonksiyon Ekle(ad: metin; fiyat: ondalık; adet: tamsayı): tamsayı
başla
  vt.Çalıştır('INSERT INTO urun (ad, fiyat, adet) VALUES (?, ?, ?)', [ad, fiyat,
      ↪  adet])
  dön vt.SonKimlik
bitir

prosedür AdetDeğiştir(no, fark: tamsayı)
başla
  vt.Çalıştır('UPDATE urun SET adet = adet + ? WHERE no = ?', [fark, no])
bitir

bitir.
```

İş katmanı kuralları uygular. HTTP durum kodu da SQL de bilmez; kural çiğnenince türü belli bir hata fırlatır:

```
birim Ürünİşleri
{ İş katmanı: kurallar burada. HTTP'yi de SQL'i de bilmez; depolama katmanını
    ↪  çağırır,
```

```
  kural çiğnenirse açıklayıcı bir hata fırlatır. }
kullan ÜrünDeposu

arayüz
  fonksiyon ÜrünListesi: liste
  fonksiyon ÜrünEkle(ad: metin; fiyat: ondalık; adet: tamsayı): sözlük
  fonksiyon Sat(no, adet: tamsayı): sözlük
  fonksiyon StokDeğeri: ondalık

uygulama

fonksiyon ÜrünListesi: liste
başla
  dön Hepsi
bitir

fonksiyon ÜrünEkle(ad: metin; fiyat: ondalık; adet: tamsayı): sözlük
değişken temiz: metin
başla
  temiz := Kırp(ad)
  eğer Uzunluk(temiz) < 2 ise fırlat Hata('Ürün adı en az 2 harf olmalı',
      ↪ 'KuralHatası')
  eğer fiyat < = 0 ise fırlat Hata('Fiyat sıfırdan büyük olmalı', 'KuralHatası')
  eğer adet < 0 ise fırlat Hata('Adet eksi olamaz', 'KuralHatası')
  eğer AdVarMı(temiz) ise fırlat Hata('Bu adda bir ürün zaten var: ' + temiz,
      ↪ 'KuralHatası')
  dön Bul(Ekle(temiz, fiyat, adet))
bitir

fonksiyon Sat(no, adet: tamsayı): sözlük
değişken ürün: herhangi
başla
  ürün := Bul(no)
  eğer ürün = boş ise fırlat Hata('Böyle bir ürün yok', 'BulunamadıHatası')
  eğer adet < 1 ise fırlat Hata('Satış adedi en az 1 olmalı', 'KuralHatası')
  eğer ürün['adet'] < adet ise fırlat Hata('Stok yetersiz: ' + ürün['adet'] + ' adet
      ↪ var', 'KuralHatası')
  AdetDeğiştir(no, -adet)
  dön ['ürün': ürün['ad'], 'adet': adet, 'tutar': ürün['fiyat'] * adet, 'kalan':
      ↪ ürün['adet'] - adet]
bitir

fonksiyon StokDeğeri: ondalık
değişken
  toplam: ondalık
  ü: sözlük
başla
  toplam := 0
  Hepsi içindeki her ü için toplam := toplam + ü['fiyat'] * ü['adet']
```

```
  dön toplam
bitir

bitir.
```

Sunum katmanı isteği iş katmanının çağrısına, iş katmanının hatasını da durum koduna çevirir:

```
program ÜçKatman
{ Sunum katmanı: HTTP isteklerini iş katmanına çevirir, sonucu JSON olarak döndürür.
  Kural bilmez, SQL bilmez; yalnızca ÜrünİşleriBirimini kullanır.
  Sunucu olarak çalıştırmak için:  yontem 10_uc_katman.y dinle }
kullan Sunucu, Veri, Sistem, ÜrünİşleriBirimi, ÜrünDeposu

değişken
  y: sözlük

{ İş katmanının hatalarını HTTP durum kodlarına çevirir }
fonksiyon HataYanıtı(h: herhangi): herhangi
başla
  eğer h.tür = 'KuralHatası' ise dön JsonYanıt(['hata': h.mesaj], 400)
  eğer h.tür = 'BulunamadıHatası' ise dön JsonYanıt(['hata': h.mesaj], 404)
  fırlat h
bitir

fonksiyon Listele(istek: sözlük): herhangi
başla
  dön JsonYanıt(['ürünler': ÜrünListesi, 'stokDeğeri': StokDeğeri])
bitir

fonksiyon YeniÜrün(istek: sözlük): herhangi
değişken v: sözlük
başla
  dene
    v := JSONÇöz(istek['gövde'])
    dön JsonYanıt(ÜrünEkle(v['ad'], v['fiyat'], v['adet']), 201)
  yakala (h)
    dön HataYanıtı(h)
  bitir
bitir

fonksiyon Satış(istek: sözlük): herhangi
başla
  dene
    dön JsonYanıt(Sat(Tamsayı(istek['parametreler']['no']),
        ↪  Tamsayı(istek['sorgu'].Al('adet', '1'))))
  yakala (h)
    dön HataYanıtı(h)
```

```
  bitir
bitir

prosedür Göster(başlık: metin; y: sözlük)
başla
  yaz(başlık + ' → ' + y['durum'] + ' ' + y['gövde'], 1)
bitir

başla
  Yol('GET', '/urunler', Listele)
  Yol('POST', '/urunler', YeniÜrün)
  Yol('POST', '/urunler/:no/sat', Satış)

  eğer Uzunluk(Argümanlar) > 0 ise
  başla
    Aç('sqlite:magaza.db')
    Dinle(8080)
    dön
  bitir

  { Argümansız çalıştırınca: bellekteki veritabanıyla, ağ olmadan bir tur }
  Aç('sqlite::bellek:')
  Göster('ekle', İstekİşle('POST', '/urunler', '{"ad": "defter", "fiyat": 12.5,
      ↪  "adet": 40}'))
  Göster('ekle', İstekİşle('POST', '/urunler', '{"ad": " kalem ", "fiyat": 4.0,
      ↪  "adet": 3}'))
  Göster('aynı ad', İstekİşle('POST', '/urunler', '{"ad": "defter", "fiyat": 1.0,
      ↪  "adet": 1}'))
  Göster('kötü fiyat', İstekİşle('POST', '/urunler', '{"ad": "silgi", "fiyat": 0.0,
      ↪  "adet": 1}'))
  Göster('sat', İstekİşle('POST', '/urunler/1/sat?adet =4'))
  Göster('stok yetersiz', İstekİşle('POST', '/urunler/2/sat?adet =5'))
  Göster('ürün yok', İstekİşle('POST', '/urunler/9/sat'))
  Göster('liste', İstekİşle('GET', '/urunler'))
bitir.
```

```
ekle → 201 {"no": 1, "ad": "defter", "fiyat": 12.5, "adet": 40}
ekle → 201 {"no": 2, "ad": "kalem", "fiyat": 4.0, "adet": 3}
aynı ad → 400 {"hata": "Bu adda bir ürün zaten var: defter"}
kötü fiyat → 400 {"hata": "Fiyat sıfırdan büyük olmalı"}
sat → 200 {"ürün": "defter", "adet": 4, "tutar": 50.0, "kalan": 36}
stok yetersiz → 400 {"hata": "Stok yetersiz: 3 adet var"}
ürün yok → 404 {"hata": "Böyle bir ürün yok"}
liste → 200 {"ürünler": [{"no": 1, "ad": "defter", "fiyat": 12.5, "adet": 36}, {"no": 2,
    ↪  "ad": "kalem", "fiyat": 4.0, "adet": 3}], "stokDeğeri": 462.0}
```

Bu ayrımın kazandırdıkları şunlardır:

- **Veritabanını değiştirmek** tek satırdır: `Aç('sqlite:magaza.db')` yerine `Aç('postgres://…')` yazmak yeterlidir. İş ve sunum katmanları değişmez.
- **Arayüzü değiştirmek** iş kurallarına dokunmaz: aynı `ÜrünİşleriBirimini` bir `Pencere` programı ya da komut satırı aracı da kullanabilir.
- **Test etmek** kolaydır: bellekteki veritabanı ve `İstekİşle` ile bütün uygulama ağ ve dosya olmadan, `yontem test` içinde sınanır. Yukarıdaki çıktı böyle üretilmiştir.

Katman sınırını dil zorlamaz: sunum katmanının `kullan ÜrünDeposu` yazıp doğrudan sorgu çalıştırmasını engelleyen bir şey yoktur (örnekte yalnızca veritabanını açmak için kullanılıyor). Sınırı korumak, her birimin `arayüz` bölümünü küçük tutmakla başlar.

10. Araçlar

Araçlar JavaScript derleyicisinin `yontem` komutuyla gelir. Komutu kurmak için `npm install -g yontem` yazmanız yeterlidir (ayrıntıları *Yöntem Kurulum Kılavuzu*'nda bulabilirsiniz).

Yeni proje: `yontem yeni`

```
yontem yeni hesap-defteri
cd hesap-defteri
yontem hesap-defteri.y
yontem test
```

Komut bir ana program, örnek bir birim, bir test ve bir `BENIOKU.md` içeren klasör oluşturur.

Test: `yontem test`

Yöntem'de test, çıktısı bilinen bir programdır. `yontem test [klasör]` her `.y` programını çalıştırır ve çıktıyı yanındaki aynı adlı `.beklenen` dosyasıyla karşılaştırır; fark varsa ilk farklı satırı gösterir. `.beklenen` yoksa programın hatasız bitmesi yeterlidir. `.girdi` dosyası varsa klavye girdisi olarak verilir.

```
program ToplaTesti
kullan Hesap                    { bir üst klasördeki Hesap.y birimi }

başla
  iddia(Topla(2, 3) = 5, 'Topla 2 + 3 için 5 vermeli')
  yaz(Topla(10, -4), 1)
bitir.
```

`iddia(koşul, mesaj)` koşul yanlışsa programı hatayla durdurur; test başarısız sayılır. Beklenen çıktıyı elle yazmak yerine `yontem test --guncelle` ile şimdiki çıktıdan üretebilirsiniz; üretilen dosyayı gözden geçirmenizi öneririz. `testler/` klasöründeki programlar bir üst klasördeki birimleri kendiliğinden bulur. Aynı komut Python yorumlayıcısında da vardır: `python3 yontem.py test`.

Katı kip: `{$katı}` ve `yontem denetle`

Yöntem tipleri olağan durumda program çalışırken denetler: `tamsayı` bir değişkene metin atayan satır, ancak o satıra gelindiğinde hata verir. Dosyanın başına `{$katı}` yazarsanız aynı denetim derleme zamanında yapılır: program hiç çalışmadan, bütün tip hataları birden listelenir.

```
program Sepet
{$katı}

değişken
  adet: tamsayı
  fiyat: ondalık
  ürün := 'defter'

fonksiyon Tutar(adet: tamsayı; fiyat: ondalık): ondalık
başla
  dön adet * fiyat
bitir

başla
  adet := '3'
  fiyat := 12.5
  eğer adet ise yaz(ürün + ': ' + Tutar(fiyat, adet), 1)
  yaz(toplm, 1)
bitir.
```

```
Tip hatası [sepet.y, satır 15] 'adet' tamsayı tipinde; metin bir değer atanamaz
Tip hatası [sepet.y, satır 17] Koşul mantıksal olmalı, tamsayı verildi
Tip hatası [sepet.y, satır 17] 'Tutar' çağrısında 1. argüman tamsayı olmalı, ondalık
    ↪ verildi
Tip hatası [sepet.y, satır 18] Tanımsız tanımlayıcı: 'toplm'
4 tip hatası bulundu; program çalıştırılmadı.
```

Katı kipte şunlar derleme hatasıdır:

- Tipi belirtilmemiş sabit: `sabit BAĞLAM = 10` yerine `sabit BAĞLAM: tamsayı = 10` yazılır.
- Tipi belirtilmemiş parametre ya da dönüş tipi belirtilmemiş fonksiyon. Değişkenlerde tip, ilk değerden çıkarılabiliyorsa yazılmayabilir (`ürün := 'defter'` bir `metin` değişkenidir); çıkarılamıyorsa belirtilmelidir.

- Tipi belli bir değeri uymayan tipte bir değişkene atamak, parametreye vermek ya da fonksiyondan döndürmek. `tamsayı`, `ondalık` beklenen yere verilebilir; tersi verilemez.
- Tipleri bildirilmiş bir alt programı eksik ya da fazla argümanla çağırmak.
- Mantıksal olmayan koşul; sayı ile metni çıkarmak ya da karşılaştırmak gibi tanımsız işlemler.
- Tanımlanmamış bir ad kullanmak.

Listelerin ve sözlüklerin eleman tipini de bildirebilirsiniz; katı kip elemanları de denetler:

```
değişken
  notlar: tamsayı listesi
  yaşlar: tamsayı sözlüğü
  tablo: ondalık listesi listesi
  ad: metin
başla
  notlar := [70, 85, 'doksan']     { hata: metin bir eleman konamaz }
  notlar.Ekle(100)
  ad := notlar[0]                  { hata: 'ad' metin tipinde; tamsayı bir değer
      ↪ atanamaz }
  yaşlar['Ali'] := 'otuz'          { hata: metin bir eleman konamaz }
```

Eleman tipi atamada, `Ekle` çağrısında, `liste[i] := değer` atamasında, `içindeki her` döngüsünün değişkeninde ve parametrelerde denetlenir; diziler için eleman tipi başa yazılır: `tamsayı dizisi`, iç içe `ondalık dizisi dizisi` (`dizisi` ile `listesi` eş anlamlıdır). Eleman tipi yazılmamış düz bir `liste` her yere verilebilir ve çalışırken denetlenir.

Boyutu `L.Uzunluk(12, 1)` ile sabitlenen listeye `Ekle` ya da `Sil` uygulamak çalışma anında hata verir; bu denetim katı kipte de çalışma anında yapılır. Boyutu değişen veri için serbest liste kullanabilirsiniz; `liste(L)` serbest bir kopya verir.

Sınıf ve kayıt tipleri de izlenir. Tipi `Daire` olan bir değişkende olmayan bir alana ya da metoda erişmek (`d.çap`), bir alana uymayan değer atamak (`d.yarıçap := 'iki'`), bir metodu ya da kurucuyu eksik veya yanlış tipte argümanla çağırmak ve bir sınıfın nesnesini ilgisiz bir sınıfın değişkenine atamak derleme hatasıdır. Alt sınıfın nesnesi üst sınıf tipindeki değişkene atanabilir (`şekil := daire`); ama o değişken üzerinden yalnızca üst sınıfın üyelerine erişilir. Alt sınıfa özgü bir üyeye ulaşmak için değeri alt sınıf tipinde bir değişkene atayabilirsiniz.

Denetim, tipi derleme anında bilinen değerleri kapsar: sabitler, tipi bildirilmiş değişkenler, parametreler, alanlar ve elemanlar, dönüş tipi bildirilmiş fonksiyonlar ve metodlar, sonucu belli olan yerleşik işlevler (`Uzunluk`, `Metin`, `Tamsayı`, `BüyükHarf`, `Ayır`, `İçerir` gibi), `Takvim`, `Desen`, `Güvenlik` ve `Ağ` birimlerinin işlevleri (`TarihYaz` bir metin, `GünFarkı` bir tamsayı, `Eşleşir` bir mantıksal değer verir) ve bunlarla kurulan ifadeler. Yerleşik işlevler ve yerleşik birimlerin bütün işlevleri için argüman sayısı da denetlenir: `TarihYap(2026, 1)` derleme hatasıdır. Sonucu argü-

mana göre değişen işlevler (`Maks`, `Yuvarla`, `Bul`) ile `Pencere`, `Meşale` gibi birimlerin döndürdüğü nesneler gibi tipi önceden bilinemeyen değerler eskisi gibi çalışırken denetlenir; katı kip çalışma zamanı denetimlerini kaldırmaz, onların önüne bir denetim daha koyar. Eleman tipi bildirimi yalnızca derleme zamanında denetlenir: çalışırken bir listeye konan elemanların tipine bakılmaz. Bir değişkeni bilerek her tipe açık bırakmak için tipini `herhangi` yazabilirsiniz.

Yönerge dosya başınadır: katı bir program katı olmayan birimleri kullanabilir, katı bir birimdeki hatalar onu kullanan program derlenirken bildirilir. Bütün projeyi katı yapmak için `yontem.json` dosyasına `"kati": true` eklemeniz yeterlidir (kurulan paketler bunun dışında kalır). Yönerge bir yorum olduğu için Python yorumlayıcısı onu görmezden gelir ve programı olağan kipte çalıştırır; derleme zamanı denetimi JavaScript derleyicisindedir.

```
yontem denetle .            # programları çalıştırmadan derler, hataları listeler
yontem denetle . --kati     # yönergesi olmayan programları da katı kipte denetler
```

`yontem denetle` sürekli tümleştirmede ve bir projeyi katı kipe geçirmeden önce ne kadar iş çıkacağını görmek için kullanışlıdır. VS Code eklentisi katı dosyalardaki tip hatalarını yazarken gösterir.

Biçimlendirme: `yontem bicimle`

```
yontem bicimle .             # klasördeki tüm .y dosyalarını düzenler
yontem bicimle . --denetle   # değiştirmeden, düzensiz dosyaları listeler
```

Biçimlendirici girintiyi programın yapısına göre yeniden kurar (düzey başına iki boşluk), satır sonu boşluklarını ve fazla boş satırları temizler. Yalnızca boşluklara dokunur: sözcükler, satır bölünmeleri ve yorumlar değişmez, bu yüzden programın anlamı değişemez. Çok satıra bölünmüş ifadelerde kendi hizalamanız korunur.

Hata ayıklama: `yontem ayikla`

Program beklediğiniz sonucu vermiyorsa araya `yaz` satırları eklemek yerine onu hata ayıklayıcıyla çalıştırabilirsiniz: program istediğiniz satırda durur, değişkenlerin o andaki değerlerine bakar, sonra satır satır ilerlersiniz. Örnek olarak notların ortalamasını bulan şu programı alalım:

```
program Ayıklama
{ Hata ayıklayıcıyı denemek için küçük bir program: notların ortalamasını bulur.
  Çalıştırmak için: yontem ayikla 07_ayiklama.y }

fonksiyon Ortalama(notlar: liste): ondalık
değişken
  toplam, n: tamsayı
başla
```

```
  toplam := 0
  notlar içindeki her n için
    toplam := toplam + n
  dön toplam / Uzunluk(notlar)
bitir

değişken
  notlar: liste
  ort: ondalık

başla
  notlar := [70, 85, 90]
  ort := Ortalama(notlar)
  yaz('Ortalama: ' + Yuvarla(ort, 1), 1)
bitir.
```

`yontem ayikla 07_ayiklama.y` programı ilk deyimde durdurur ve komut bekler. Aşağıdaki oturumda önce bir satır ilerliyor, sonra `Ortalama` fonksiyonunun içine giriyor, döngünün içine yalnızca `n > 80` olduğunda duran bir kesme noktası koyuyor ve durduğu yerde değişkenlere bakıyoruz:

```
$ yontem ayikla 07_ayiklama.y
Yöntem hata ayıklayıcı — 07_ayiklama.y (komutlar için: yardım)
→   20    notlar := [70, 85, 90]
(ayıkla) sonraki
→   21    ort := Ortalama(notlar)
(ayıkla) adım
Ortalama içinde, 07_ayiklama.y:9
→    9    toplam := 0
(ayıkla) kes 11 eğer n > 80
Kesme noktası kondu: 07_ayiklama.y:11 (eğer n > 80)
(ayıkla) devam
Kesme noktası: 07_ayiklama.y:11
Ortalama içinde, 07_ayiklama.y:11
→●  11      toplam := toplam + n
(ayıkla) değişkenler
Ortalama:
  notlar = [70, 85, 90]
  sonuç = 0.0
  toplam = 70
  n = 85
Program:
  ort = 0.0
(ayıkla) yaz toplam + n
155
(ayıkla) yığın
→ #0  Ortalama  07_ayiklama.y:11
  #1  program Ayıklama  07_ayiklama.y:21
(ayıkla) sil 11
```

```
Kesme noktası kaldırıldı: 07_ayiklama.y:11
(ayıkla) çık
→   22    yaz('Ortalama: ' + Yuvarla(ort, 1), 1)
(ayıkla) yaz ort
81.66666666666667
(ayıkla) devam
Ortalama: 81.7
Program bitti (çıkış kodu 0).
```

Okun (→) gösterdiği satır henüz çalışmamıştır; sıradaki deyimdir. ● kesme noktasını gösterir.

Komut	Ne yapar
`adım` (`a`)	Sonraki deyime geçer; satırda bir çağrı varsa içine girer
`sonraki` (`s`)	Sonraki satıra geçer; çağrıların içine girmez
`çık` (`ç`)	Bulunulan alt programdan çıkana kadar çalışır
`devam` (`d`)	Bir kesme noktasına ya da programın sonuna kadar çalışır
`kes 12` (`k`)	Kesme noktası koyar. Başka bir birimde: `kes Birim:12`. Koşullu: `kes 12 eğer i > 3`
`sil 12`	Kesme noktasını kaldırır; `sil hepsi` hepsini
`yaz ifade` (`y`)	İfadeyi değerlendirir: `y toplam * 2`. Atama değişkeni değiştirir: `y x := 5`
`değişkenler` (`dğ`)	Bulunulan alt programın, nesnenin ve programın değişkenlerini listeler
`yığın` (`yğ`)	Buraya hangi çağrılarla gelindiğini gösterir; `yukarı` ve `aşağı` çağıranın değişkenlerine geçirir
`liste` (`l`)	Çevredeki kaynak satırlarını gösterir
`hatalar kapat`	Çalışma hatasında durmayı kapatır (`hatalar aç` geri açar)
`bitir` (`q`)	Programı sonlandırır

Boş satır son adımlama komutunu yineler; komutlar Türkçe karaktersiz de yazılabilir (`adim`, `cik`, `degiskenler`). Program bir çalışma hatası verdiğinde ayıklayıcı hatanın oluştuğu satırda durur: değişkenler hâlâ yerindedir, hataya neyin yol açtığına bakabilirsiniz. Hata bir `dene … yakala` bloğunda yakalanacak olsa da durur; bunu istemiyorsanız `hatalar kapat` yazabilirsiniz. Programın `oku` çağrıları ile ayıklayıcı komutları aynı girdiyi paylaşır: program girdi beklerken yazdığınız satır programa gider.

Hata ayıklayıcı JavaScript derleyicisinin aracıdır; ayıklama kipinde derlenen program aynı çıktıyı verir, yalnızca biraz daha yavaş çalışır.

Paketler: `yontem paket`

Başkasının yazdığı birimleri projenize paket olarak kurarsınız. Paket, içinde birim dosyaları olan bir klasördür; merkezî bir paket deposu yoktur, paket bulunduğu yerden kurulur:

```
yontem paket ekle github:kullanici/depo@v1.2.0     # GitHub deposu (etiket, dal ya da
    ↪ commit)
yontem paket ekle https://ornek.org/araclar.zip    # arşiv (.tar.gz, .zip) ya da tek .y
    ↪ dosyası
yontem paket ekle ../ortak-birimler                # yerel klasör
```

`ekle` paketi `paketler/` klasörüne indirir ve iki dosyayı günceller. `yontem.json` projenizin bildirimidir: adı, sürümü ve doğrudan kullandığı paketler.

```
{
  "ad": "hesap",
  "surum": "0.1.0",
  "bagimliliklar": {
    "Selam": "github:kullanici/selam@v1.2.0"
  }
}
```

`yontem.kilit.json` ise kurulan her paketin (paketlerin kendi bağımlılıkları dahil) tam olarak nereden geldiğini ve içeriğinin SHA-256 özetini tutar. Kurulduktan sonra paketin birimleri kendi birimleriniz gibi kullanılır: `kullan Selam`. Program, bulunduğu klasörden yukarı doğru `yontem.json` içeren ilk klasörün paketlerini görür; birimler paketin kökünde ya da `kutuphane/` klasöründe aranır. JavaScript derleyicisi ve Python yorumlayıcısı aynı kuralı uygular.

Komut	Ne yapar
`yontem paket ekle <kaynak>`	Paketi kurar ve `yontem.json` dosyasına yazar. Ad paketten gelir; `--ad Ad` ile değiştirilir
`yontem paket kur`	`yontem.json` dosyasındaki paketleri kurar, içeriği kilit dosyasıyla doğrular
`yontem paket kaldir <ad>`	Paketi ve başka hiçbir paketin istemediği bağımlılıklarını kaldırır
`yontem paket guncelle [ad]`	Paketi (ya da hepsini) kaynağından yeniden indirir, kilit dosyasını yeniler
`yontem paket liste`	Kurulu paketleri, sürümlerini ve sağladıkları birimleri gösterir

`yontem.json` ve `yontem.kilit.json` dosyalarını sürüm denetimine koymanızı, `paketler/` klasörünü ise dışarıda bırakmanızı öneririz: projeyi alan kişi `yontem paket kur` ile aynı paketleri kurar. İndirilen içerik kilit dosyasındaki özetle aynı

değilse kurulum durur; kaynağın değiştiğini siz `guncelle` ile onaylamadıkça yeni içerik projenize girmez. Bu yüzden kaynağı dal adıyla değil etiketle (`@v1.2.0`) vermek en sağlıklısıdır.

Kurulum yalnızca dosya kopyalar, paketteki hiçbir kod kurulurken çalışmaz; arşivlerde klasör dışına çıkan yollar reddedilir ve indirme yalnızca HTTPS ile yapılır. Yine de paket, programınızla birlikte çalışacak koddur: güvenmediğiniz bir kaynaktan paket kurmamanızı, kurduğunuz paketin birimlerini de okumanızı öneririz.

Kendi paketinizi yayımlamak için birimlerinizi bir Git deposunun köküne (ya da `kutuphane/` klasörüne) koyup sürümü etiketlemeniz yeterlidir. Depoya bir `yontem.json` eklerseniz paketin adı ve sürümü oradan okunur; paketiniz başka paketlere dayanıyorsa `bagimliliklar` altında yazdıklarınız onu kuranlara kendiliğinden kurulur. İki paket aynı adlı paketi farklı kaynaklardan isterse kurulum durur ve hangisini seçeceğinizi sorar.

Paketleme: `yontem paketle`, `yontem derle`, `yontem gomulu`

Komut	Ürettiği
`yontem paketle program.y [--mobil] [--ad "Ad"]`	`Pencere/Mobil` programından web uygulaması; `--mobil` ile telefona kurulabilir
`yontem derle program.y -o program.js`	Node.js ile çalışan JavaScript dosyası
`yontem gomulu program.y`	ESP32 için Arduino/PlatformIO projesi

VS Code eklentisi

Depodaki `araclar/vscode/yontem-dili-0.6.0.vsix` dosyasını `code --install-extension` ile ya da Eklentiler → **VSIX'ten Yükle** ile kurabilirsiniz. Eklenti sözdizimini renklendirir, `eğer`, `için`, `fonksiyon`, `sınıf` gibi kod parçacıkları sunar, `F5` ile etkin programı çalıştırır ve **Belgeyi Biçimlendir** (`Shift+Alt+F`) komutunu `yontem bicimle`'ye bağlar. Yazarken sözdizimi hatalarını ve bulunamayan birimleri altını çizerek gösterir, hiçbir yerde tanımlı olmayan adlar için yazım hatası uyarısı verir; dosyadaki adları, kullanılan birimlerin üyelerini ve yerleşikleri tamamlar (`Güvenlik.` yazınca yalnızca o birimin üyeleri gelir); bir adın üzerine gelince imzasını gösterir ve `F12` ile tanımına gider.

Hata ayıklayıcı da eklentidedir: satır numarasının soluna tıklayarak kesme noktası koyabilir, `F6` ile (ya da **Çalıştır ve Hata Ayıkla** panelinden) ayıklamayı başlatabilirsiniz. Program kesme noktasında durduğunda değişkenler, çağrı yığını ve izleme ifadeleri yan panelde görünür; listeler, sözlükler ve nesneler açılarak incelenir, adımlama düğmeleri `adım`, `sonraki` ve `çık` komutlarının karşılığıdır. Kesme noktasına sağ tıklayıp koşul yazabilirsiniz. VS Code'da programın `oku` çağrıları klavyeden okuyamaz; girdi gerekiyorsa `launch.json` içinde `"girdi": "girdi.txt"` ile bir dosya verebilirsiniz.

`yontem` komutu `PATH`'te değilse eklentinin `yontem.komut` ayarına tam komutu yazabilirsiniz.

Başka düzenleyiciler: `yontem dil-sunucusu`

Aynı özellikler bir dil sunucusu olarak da sunulur. Language Server Protocol destekleyen düzenleyicilerde (Neovim, Helix, Emacs, Sublime Text...) `.y` dosyaları için sunucu komutu olarak `yontem dil-sunucusu` (kısa adı `yontem lsp`) vermeniz yeterlidir; sunucu standart girdi ve çıktı üzerinden konuşur.

11. Birimleri birlikte kullanmak

Birimler birbirinden bağımsızdır ama asıl güçleri birlikte kullanıldıklarında ortaya çıkar. Aşağıdaki program bir sunucu günlüğünü inceler: başarısız girişleri `Desen` ile ayıklar, ilk ve son deneme arasındaki süreyi `Takvim` ile hesaplar, e-posta adreslerini `Güvenlik` ile maskeler ve raporu imzalar.

```
program GirişRaporu
{ Üç birim birlikte: bir sunucu günlüğünden başarısız girişler Desen ile ayıklanır,
  Takvim ile zaman farkı hesaplanır, Güvenlik ile e-postalar maskelenir ve rapor
      ↪ imzalanır. }
kullan Desen, Takvim, Güvenlik

sabit
  GÜNLÜK: metin = '2026-11-10 08:59:40 giriş-başarısız ayse@ornek.de' + #10 +
           '2026-11-10 09:00:05 giriş-başarısız ayse@ornek.de' + #10 +
           '2026-11-10 09:00:31 giriş-başarılı mehmet@ornek.de' + #10 +
           '2026-11-10 09:01:02 giriş-başarısız ayse@ornek.de' + #10 +
           '2026-11-10 14:30:00 giriş-başarısız can@ornek.de'
  KALIP: metin = '^(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}) giriş-başarısız (\S+)$'
  RAPOR_ANAHTARI: metin = 'rapor-imza-anahtari'

değişken
  ilk, son, sayı: sözlük
  g: herhangi
  an: tamsayı
  rapor, kim, satır: metin

başla
  ilk := [:]; son := [:]; sayı := [:]
  Ayır(GÜNLÜK, #10) içindeki her satır için
```

```
  başla
    g := Gruplar(satır, KALIP)
    eğer g <> boş ise
    başla
      an := TarihOku(g[1])
      kim := g[2]
      eğer değil sayı.İçerir(kim) ise
      başla
        sayı[kim] := 0
        ilk[kim] := an
      bitir
      sayı[kim] := sayı[kim] + 1
      son[kim] := an
    bitir
  bitir

  rapor := ''
  Sırala(Anahtarlar(sayı)) içindeki her kim için
  başla
    satır := Maskele(kim, 10) + ': ' + sayı[kim] + ' başarısız giriş'
    eğer sayı[kim] > = 3 ise
      satır := satır + ', ' + (son[kim] - ilk[kim]) + ' saniye içinde — ŞÜPHELİ (' +
          ↪ TarihYaz(son[kim], 'SS:DD') + ')'
    yaz(satır, 1)
    rapor := rapor + satır + #10
  bitir
  yaz('Rapor imzası: ' + AltMetin(İmza(RAPOR_ANAHTARI, rapor), 1, 16) + '…', 1)
bitir.
```

```
***e@ornek.de: 3 başarısız giriş, 82 saniye içinde — ŞÜPHELİ (09:01)
**n@ornek.de: 1 başarısız giriş
Rapor imzası: 1d84ef2e1d17c6bf…
```

Raporu alan kişi aynı anahtarla `İmzaDoğru(RAPOR_ANAHTARI, rapor, imza)` çağırarak metnin yolda değişmediğinden emin olur. Programın yapısı tipiktir: her birim kendi işini yapar, aralarındaki veri sıradan metin, tamsayı ve sözlüklerdir.

Sırada ne var?

- Örnekleri oyun alanında açıp değiştirebilirsiniz: `Pencere` örnekleri orada canlı çalışır.
- Kendi biriminizi yazabilirsiniz: *Programlamaya Giriş* kitabının 11. bölümü birimleri anlatır. Bir `.y` dosyasını programınızın yanına koymanız `kullan` ile çağırmak için yeterlidir.
- Programınıza `yontem yeni` ile başlayabilir, `yontem test` ile sınayabilir, `yontem bicimle` ile düzenleyebilir, takıldığınız yerde de `yontem ayikla` ile adım adım izleyebilirsiniz.

- Birimlerinizi başkalarıyla paylaşabilirsiniz: bir Git deposuna koyup etiketlemeniz, `yontem paket ekle` ile kurulabilmeleri için yeterlidir.