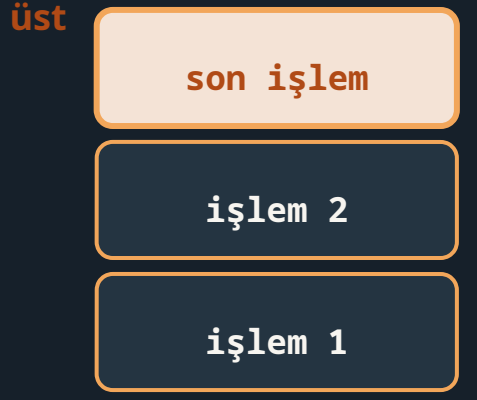
VERİ YAPILARI · 4. HAFTA

Yığın

stack



GEÇEN HAFTADAN ÜÇ SORU

Önce kısa bir hatırlama yapalım

1. Bağlı listede başa ekleme neden O(1)?
2. 500 numaralı düğüme neden baştan gideriz?
3. Bir düğümü silerken neden önceki düğümü tutarız?

Cevapları bugün yığın üzerinde göreceğiz

Bağlı listeden yeni bir kullanım çıkaracağız.

GEÇEN HAFTANIN BIRAKTIĞI SORU

Geri al tuşu en son ne yaptığımızı nereden biliyor?

Cevapta işlemlerin sırasını koruyan bir yapı var.

TEMEL FİKİR

Yığın nedir?

Yığın, yalnızca üstündeki elemana eriştiğimiz bir veri yapısıdır.



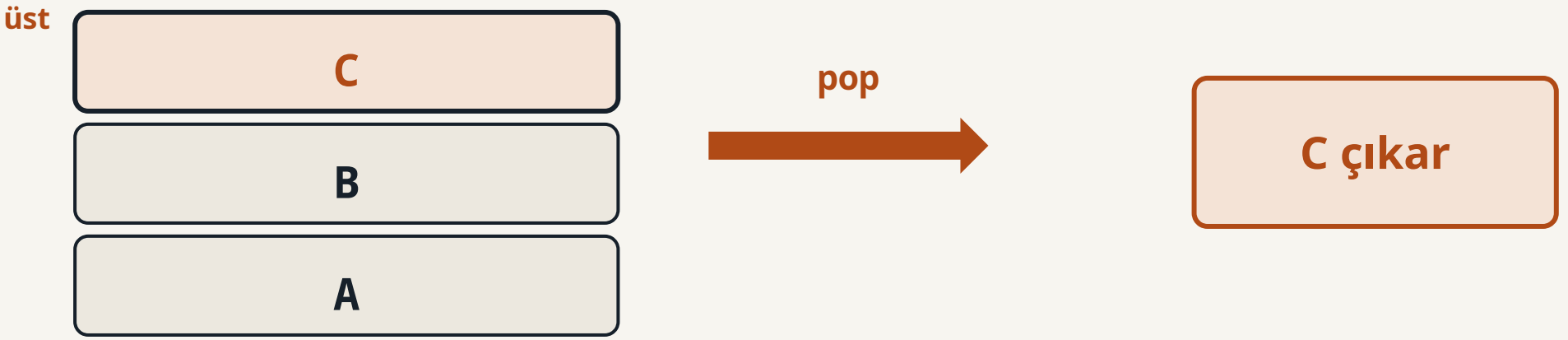
Yığının çalışma noktası

Yeni eleman buraya gelir.
Çıkan eleman da buradan ayrılır.

ÇALIŞMA İLKESİ

Son giren ilk çıkar

LIFO, last in first out ifadesinin kısaltmasıdır.



C en son girdiği için ilk çıkar. A ve B yerinde kalır.

BUGÜNÜN ANA SORUSU

En son yaptığımız işlemi nasıl önce geri alırız?

İşlemleri üstte tutan yığın, geri alma sırasını hazırlar.

DÖRT TEMEL İŞLEM

Yığının işlemleri

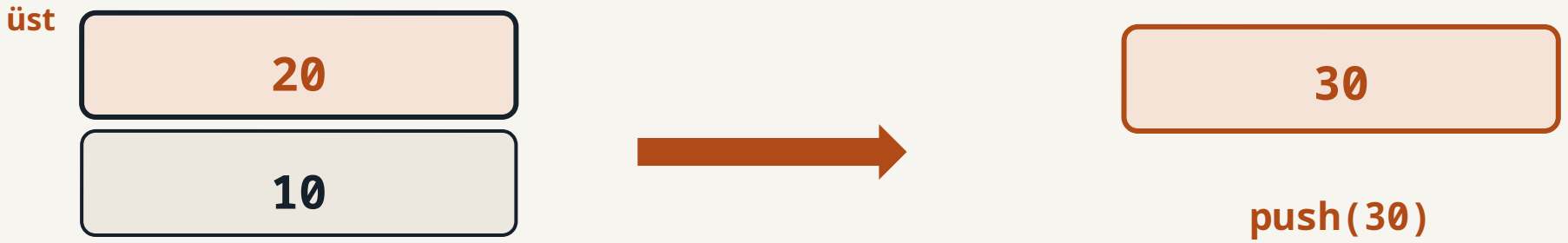
Her işlem üstteki elemana göre tanımlanır.

`push(x)`	x'i üste koyar	ekleme
`pop()`	üstü çıkarır ve döndürür	çıkarma
`peek()`	üstü silmeden gösterir	bakma
`bos_mu()`	yığın boş mu diye bakar	kontrol

PUSH

Push yeni elemanı üste koyar

Bağlantı ya da listenin sonu yığının üstüdür.



Yeni eleman üst olur. Eski üst 20 aşağıda kalır.

POP

Pop üst elemanı çıkarır

Çıkarmadan önce üstteki değeri saklarız.

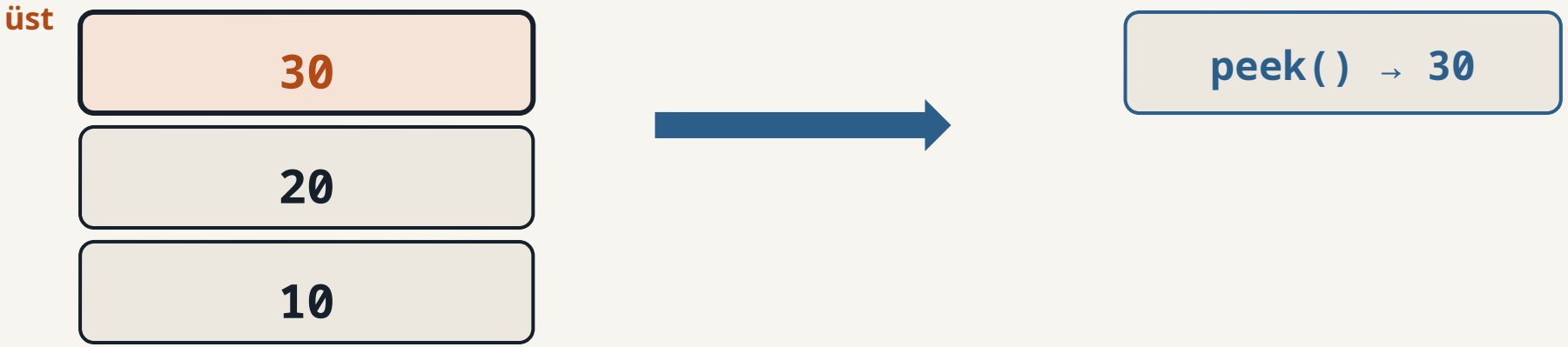


Sonra üst 20 olur.

PEEK VE SINIR DURUMU

Peek silmeden bakar

Boş yığında pop ve peek için bir karar gerekir.



Boş yığında pop ve peek IndexError üretir.

PYTHON LİSTESİ İLE

Liste, yığın için yeterli olabilir

Listenin sonunu üst kabul edersek üç işlem doğrudan yazılır.

```
class Yigin:                          Yığının tarifi
    def __init__(self):               Yığın kurulunca çalışır
        self.veriler = []             Elemanlar boş bir listede durur

    def push(self, veri):             Üste ekleme işlemi
        self.veriler.append(veri)     Listenin sonu üst, sona ekler, O(1) amortize

    def pop(self):                    Üstten çıkarma işlemi
        if not self.veriler:          Liste boş mu diye bakar
            raise IndexError("yığın boş")   Boşsa hata verir
        return self.veriler.pop()     Sondakini çıkarıp döndürür, O(1) amortize
```

LİSTE İLE MALİYET

Üstü listenin sonunda tutmak gerekir

Listenin başında yapılan çıkarma kaydırma başlatır.

Üstü sonda tut	`append`	`O(1) amortize`	yeni eleman sona gelir
Üstü sonda tut	`pop`	`O(1) amortize`	son eleman çıkar
Üstü başta tut	`insert(0, x)`	`O(n)`	elemanlar sağa kayar
Üstü başta tut	`pop(0)`	`O(n)`	elemanlar sola kayar

Yığın için seçtiğimiz uç, maliyeti belirler.

BAĞLI LİSTE İLE

Bağlı listede üst baştır

Yeni düğümü başa bağlamak push işlemini sabit tutar.

```
class Dugum:                          Düğümün tarifi
    def __init__(self, veri):         Düğüm kurulunca çalışır
        self.veri = veri              Değeri saklar
        self.sonraki = None           Altındaki düğüm henüz yok

class BagliYigin:                     Bağlı yığının tarifi
    def __init__(self):               Yığın kurulunca çalışır
        self.ust = None               Başta üst yok, yığın boş

    def push(self, veri):             Üste ekleme işlemi
        yeni = Dugum(veri)            Yeni düğümü oluşturur
        yeni.sonraki = self.ust       Yeni düğüm eski üstü gösterir
        self.ust = yeni               Üst artık yeni düğüm, O(1)
```

Bağlantı sırası

Yeni düğüm oluşturulur.

Eski üst gösterilir.

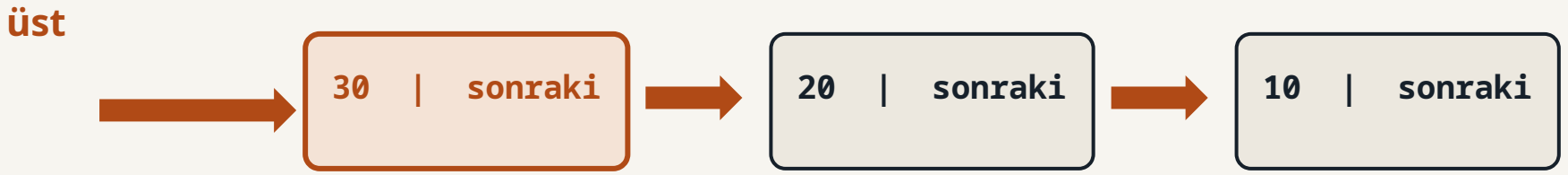
Üst yeni düğüm olur.

Push O(1).

BAĞLI YIĞIN

Bağlı yığında yalnızca üstü izleriz

Yığın için listenin tamamını dolaşmaya gerek yoktur.



`üst = 30`

Her push ve pop bağlantı değiştirir.

BAĞLI YIĞINDA POP

Pop üstü bir sonraki düğüme taşır

Eski üstü saklayıp bağlantıyı bir adım ilerletiriz.

```
def pop(self):
    if self.ust is None:
        raise IndexError("yığın boş")
    silinen = self.ust.veri
    self.ust = self.ust.sonraki

    return silinen
```

Üstten çıkarma işlemi

Yığın boş mu diye bakar

Boşsa hata verir

Üstteki değeri saklar

Üst bir alttaki düğüm olur, O(1)

Saklanan değeri döndürür



üst artık 20

İKİ GERÇEKLEŞTİRME

Liste ve bağlı liste aynı ilkeyi taşır

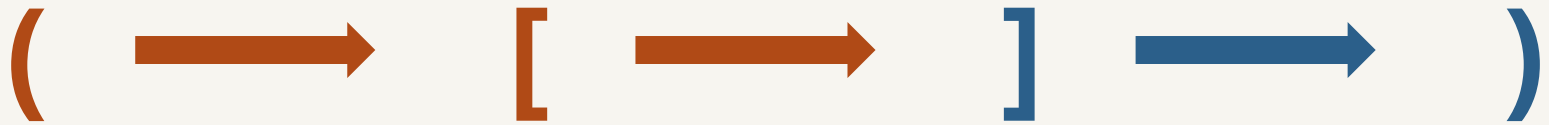
İkisi de üstten ekler ve üstten çıkarır.

Üstü temsil eden yer	listenin sonu	bağlı listenin başı
push	`append`	`başa bağla`
pop	`pop`	`üstü ilerlet`
peek	`[-1]`	`ust.veri`
Bellek	`bitişik alan`	`ayrı düğümler`

UYGULAMA

Parantez denetimi yığınla yapılır

Kapanan parantez, en son açılan parantezle eşleşmelidir.



[açıldıktan sonra önce] kapanır. Sonra (kapanır.

PARANTEZ KODU

Kapanış parantezi üstü kontrol eder

Her kapanış işareti, yığının üstündeki açılışla eşleşmelidir.

```
def parantezler_dengeli_mi(metin):                  Metnin dengeli olup olmadığını söyler
    yigin = []                                      Açılışlar için boş yığın
    eslesme = {')': '(', ']': '[',                  Her kapanışın açılış eşi
               '}': '{'}                            Sözlüğün devamı
    for karakter in metin:                          Metni karakter karakter gezer
        if karakter in '([{':                       Açılış mı diye bakar
            yigin.append(karakter)                  Açılışı üste koyar
        elif karakter in ')]}':                     Kapanış mı diye bakar
            if not yigin or yigin.pop() != eslesme[karakter]:   Yığın boşsa ya da üst eşleşmezse
                return False                        Dengesiz, hemen False
    return not yigin                                Sonda yığın boşsa dengeli
```

PARANTEZ ÖRNEKLERİ

Yığın her örnekte son açılışı izler

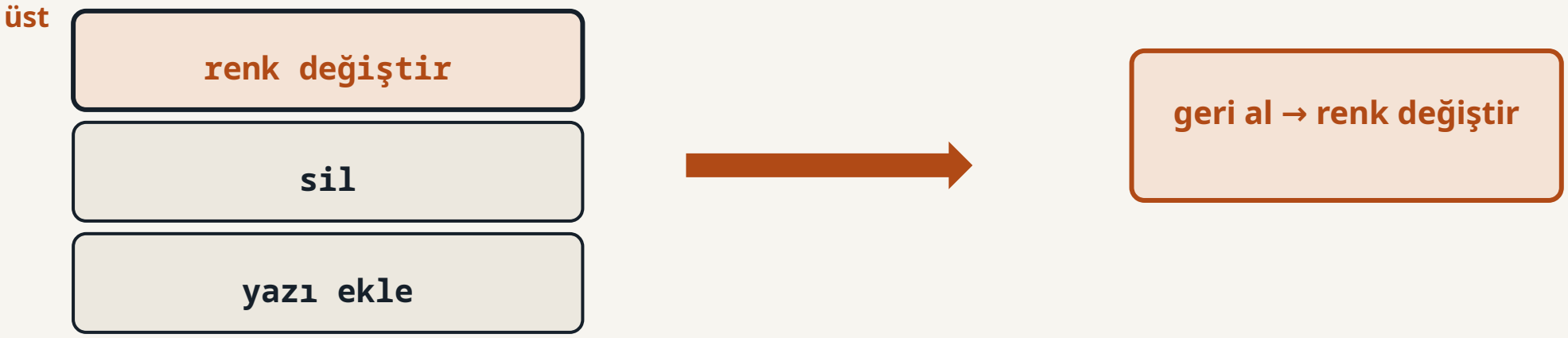
Kapanış sırası bozulduğunda denetim durur.

`( [ ] )`	**dengeli**	önce] sonra)
`{ [ ( ) ] }`	**dengeli**	içten dışa kapanır
`( [ ) ]`	**dengesiz**	) için (gerekir, üstte [var
`( (`	**dengesiz**	açık parantez kaldı

GERİ ALMA

Geri alma işlemleri yığında tutulur

Yeni işlem üste gelir. Geri al, üstteki işlemi çıkarır.

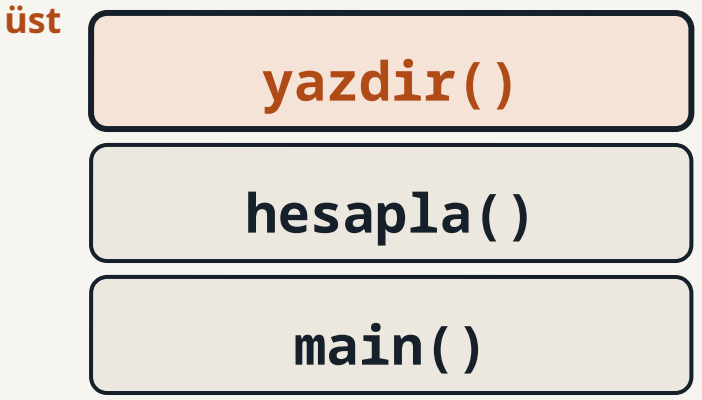


Bir sonraki geri alma silme işlemini çıkarır.

FONKSİYON ÇAĞRILARI

Python da çağrı yığını kullanır

Bir fonksiyon başka bir fonksiyonu çağırınca yeni çağrı üste gelir.



yazdir() bitince üstten çıkar.

Özyineleme konusunda bu yapıyı yeniden göreceğiz.

SIK YAPILAN HATA

Yığının üstünü listenin başında tutmak

Kod çalışır fakat insert(0, x) ve pop(0) bütün elemanları kaydırır.

```
def pop(self):                          Üstü başta tutan hatalı pop
    if not self.veriler:                Liste boş mu diye bakar
        raise IndexError("yığın boş")   Boşsa hata verir
    return self.veriler.pop(0)          Baştan çıkarır, herkes kayar
```

pop(0)

O(n)

Üstü listenin sonunda tutmak, amortize O(1) pop seçeneğini korur.

İŞLEM MALİYETLERİ

Yığında temel işlemler sabit kalabilir

Maliyeti seçtiğimiz üst konumu ve gerçekleştirmeyi belirler.

push	liste sonu	`O(1) amortize`
pop	liste sonu	`O(1) amortize`
peek	liste sonu	`O(1)`
push	bağlı liste başı	`O(1)`
pop	bağlı liste başı	`O(1)`
peek	bağlı liste başı	`O(1)`

YAPI SEÇİMİ

Yığını hangi uygulamada seçeriz?

İşlemlerin sırası ve erişim ihtiyacı seçimi belirler.

Liste ile yığın

Küçük ve geçici veriler
Python listesi zaten hazır
Üstü listenin sonunda tut

Bağlı liste ile yığın

Başa ekleme ve çıkarma sık
Başa işlem için uygun
Üst referansı başı gösterir

Her iki seçimde de ilke LIFO olarak kalır.

TAM UYGULAMA

Yığın sınıfının tamamı

Kodda boş yığın ve üstten çalışma durumlarını açıkça tutuyoruz.

```
class Yigin:                              Yığının tarifi
    def __init__(self):                   Yığın kurulunca çalışır
        self.veriler = []                 Elemanlar boş listede durur

    def push(self, veri):                 Üste ekleme işlemi
        self.veriler.append(veri)         Sona ekler, O(1) amortize

    def pop(self):                        Üstten çıkarma işlemi
        if not self.veriler:              Yığın boş mu diye bakar
            raise IndexError("yığın boş") Boşsa hata verir
        return self.veriler.pop()         Sondakini çıkarıp döndürür, O(1) amortize

    def peek(self):                       Üste bakma işlemi
        if not self.veriler:              Yığın boş mu diye bakar
            raise IndexError("yığın boş") Boşsa hata verir
        return self.veriler[-1]           Sondakini silmeden döndürür
```

KODDA KULLANDIĞIMIZ YAPI

Notebook'ta yığını adım adım kuracağız

Önce sınıfı, sonra işlemleri, ardından parantez örneğini çalıştıracağız.

1

Yığın sınıfı

`veriler = []`

2

push ve pop

`üstte çalışır`

3

peek ve boş durum

`hata davranışı`

4

parantez denetimi

`LIFO'yu kullanır`

TAHMİN ALIŞTIRMASI

İlk üç fonksiyonun Big O değeri

Kodu çalıştırmadan önce maliyeti kağıda yazın.

`push_degeri(yigin, deger)`	O( )
`pop_degeri(yigin)`	O( )
`ustu_goster(yigin)`	O( )

İpucu Üstü bulmak için kaç elemana bakılıyor?

TAHMİN ALIŞTIRMASI

Son üç fonksiyonun Big O değeri

En kötü durumda yığın boyunca dolaşan kodu ayırın.

`yigin_degerlerini_say(yigin)`	`O(  )`
`parantezleri_kontrol_et(metin)`	`O(  )`
`yigini_listele(yigin)`	`O(  )`

Cevapları notebook'ta bağlantı ve döngüler üzerinden kontrol edeceğiz.

DERSİN TEK CÜMLESİ

Yığın, son giren elemanı önce çıkarır.

Üstte yapılan birkaç işlemle LIFO sırasını korur.

GELECEK HAFTA

Baştan çıkarmak dizide yavaştı. Kuyruk bunu nasıl hızlı yapıyor?

kuyruk · queue