



YZM 2116- VERİ YAPILARI

DERŞ#3: YIĞIN VERİ YAPISI

İÇERİK

Bu bölümde,

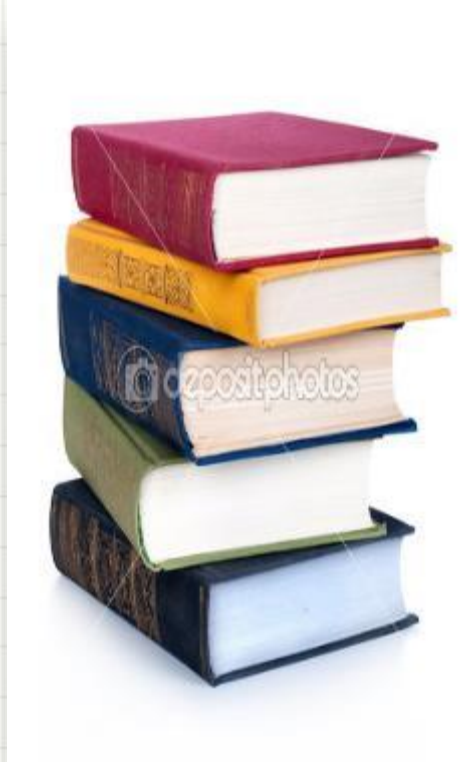
- Stack (Yığın, Yığıt) Veri Yapısı
- Stack Çalışma Şekli
- Stack Dizi Implementasyon
- Stack Uygulamaları

konularına değinilecektir.

Stack Tanımı

- Stack, doğrusal artan bir veri yapısı olup;
- Insert (**push**) ve Delete (**pop**) işlemleri,
 - Listenin sadece **“top”** adı verilen bir ucunda yani stack’in en üstünden gerçekleştirilir.
- Bu nedenle stack
 - **Son Giren İlk Çıkar (Last In First Out - LIFO)** mantığı ile işleyen bir veri yapısıdır.

Stack Tanımı (devam...)



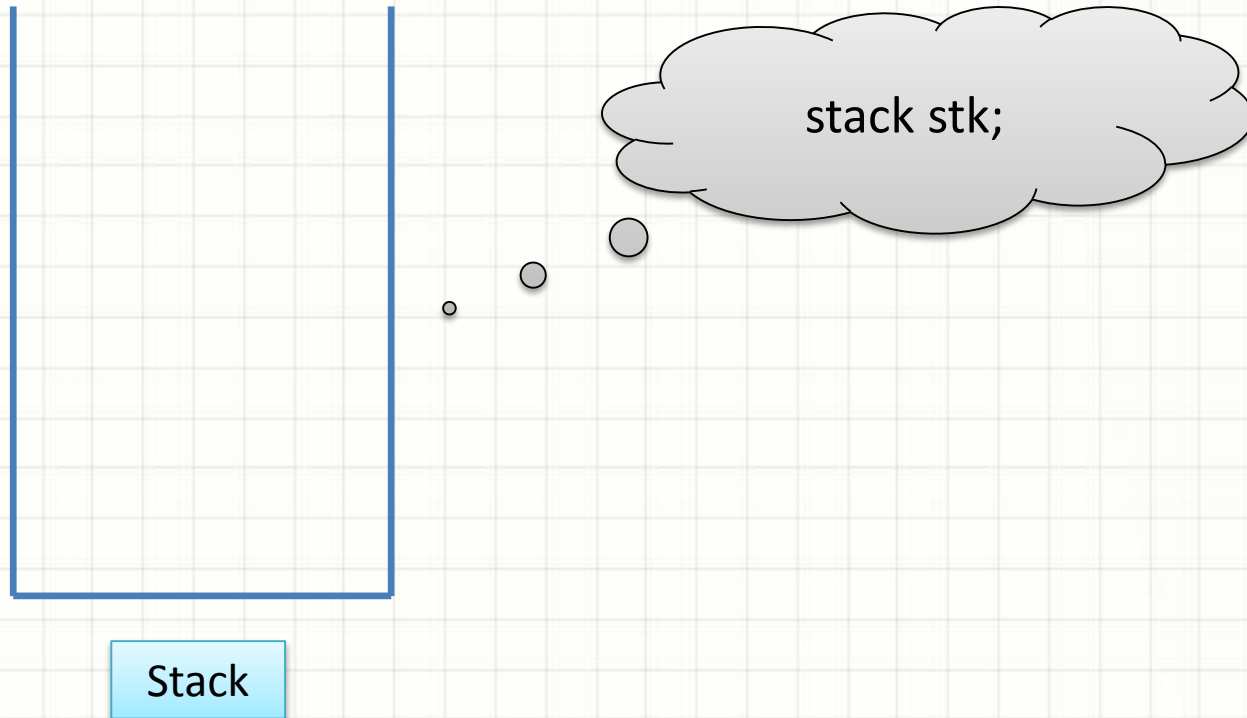
Stack ADT Interface

```
public interface IStack
{
    void Push(object item);
    object Pop();
    object Peek();
    bool IsEmpty();
    int Top { get; set; }
}
```

- **Push**: Elemanı yığıtın üstüne ekle
- **Pop**: Yığıtın üstündeki elemanı çıkart
- **Peek**: En üstteki elemanı oku
- **IsEmpty**: Yığıt boş mu?

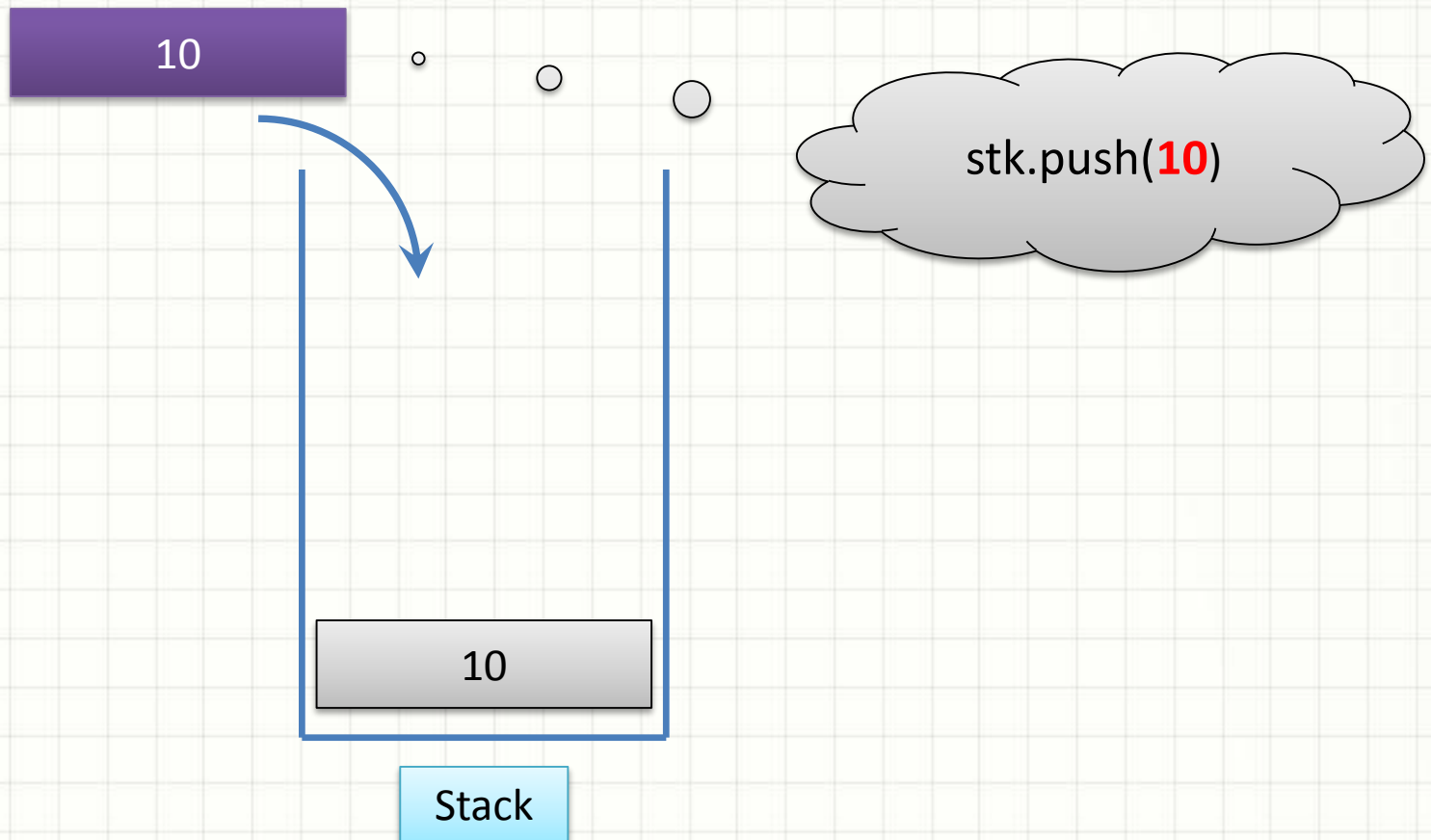
Stack Çalışma Şekli

- Stack sınıfından tanımlanmış **stk** isimli bir yığıt olsun.
- İlk aşamada **yığıt boş**.



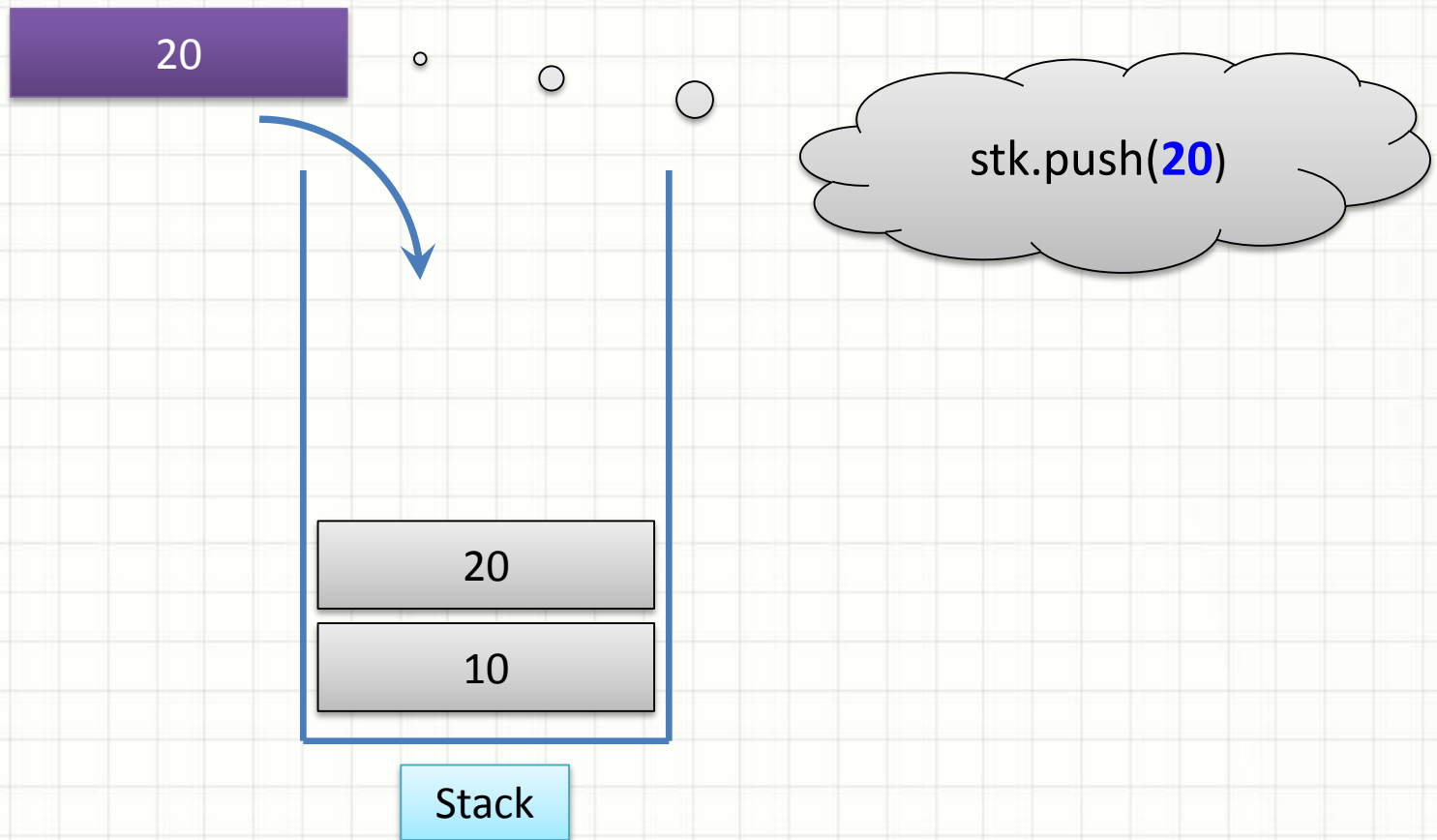
Stack Çalışma Şekli (devam...)

- Push ile yığıtın üstüne yeni bir eleman ekliyoruz.



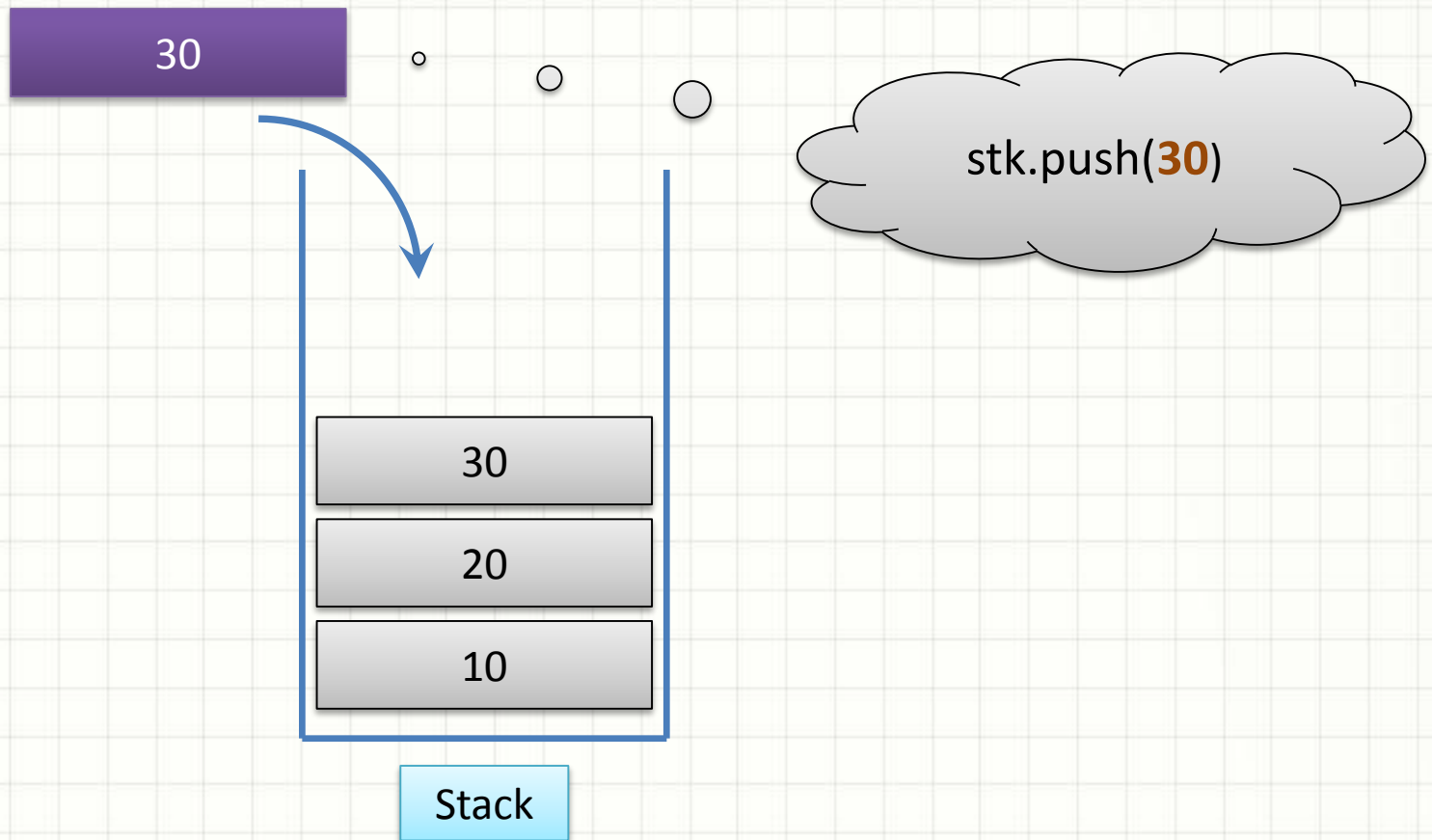
Stack Çalışma Şekli (devam...)

- Push ile yığıtta yeni bir eleman ekliyoruz.



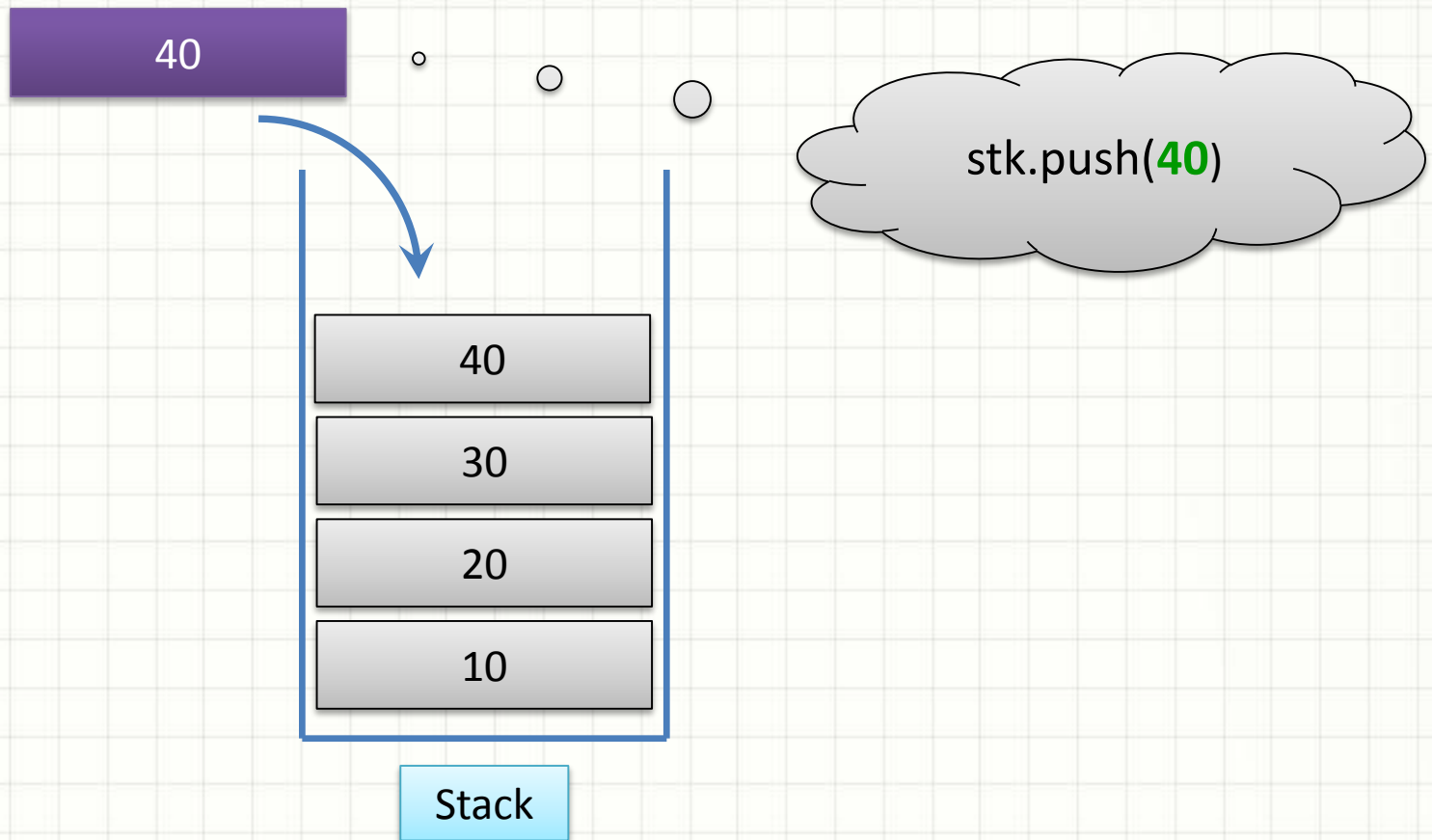
Stack Çalışma Şekli (devam...)

- Push ile yığıtta yeni bir eleman ekliyoruz.



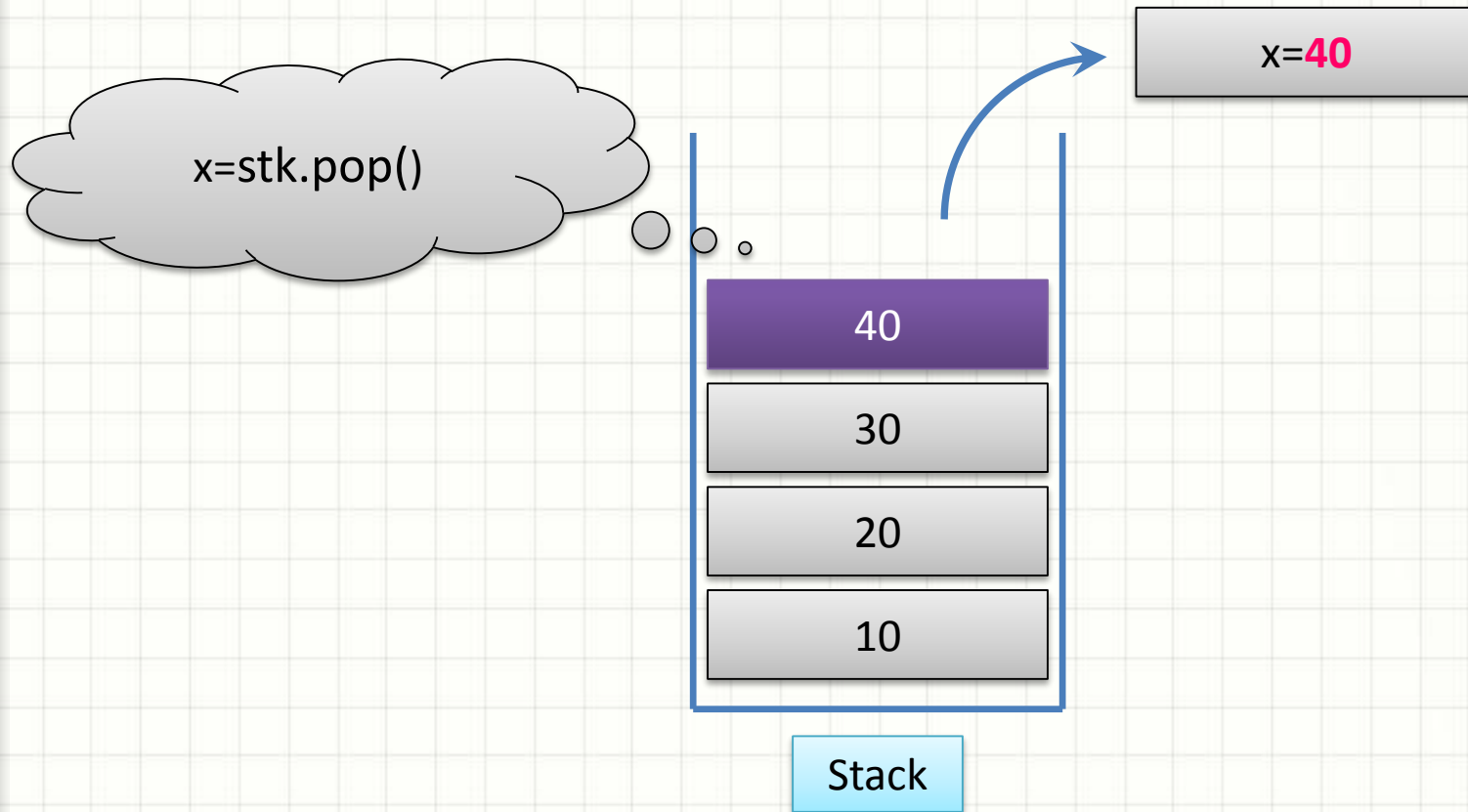
Stack Çalışma Şekli (devam...)

- Push ile yığıtta yeni bir eleman ekliyoruz.



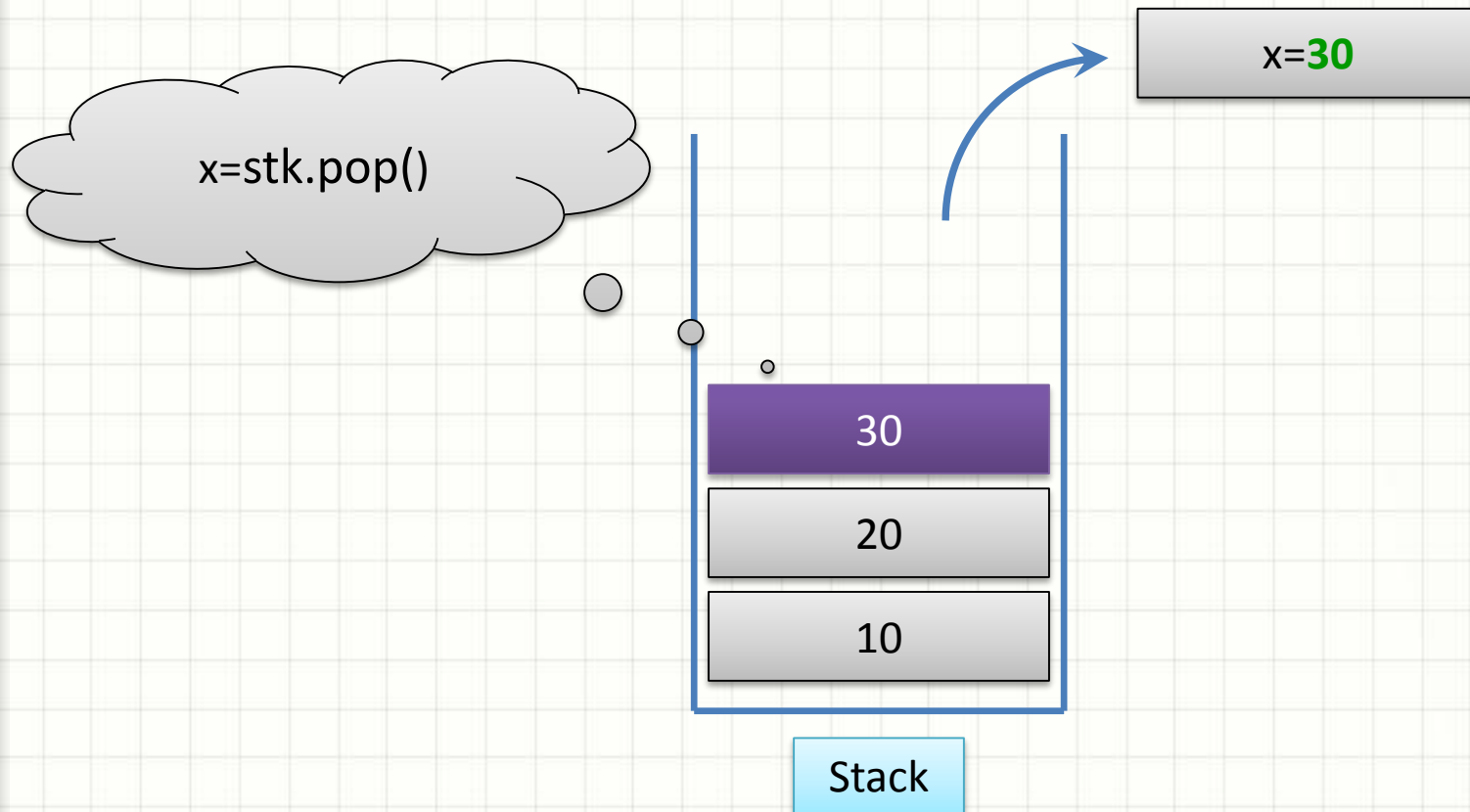
Stack Çalışma Şekli (devam...)

- Pop ile yığıtın üstünden bir eleman çıkartıyoruz.



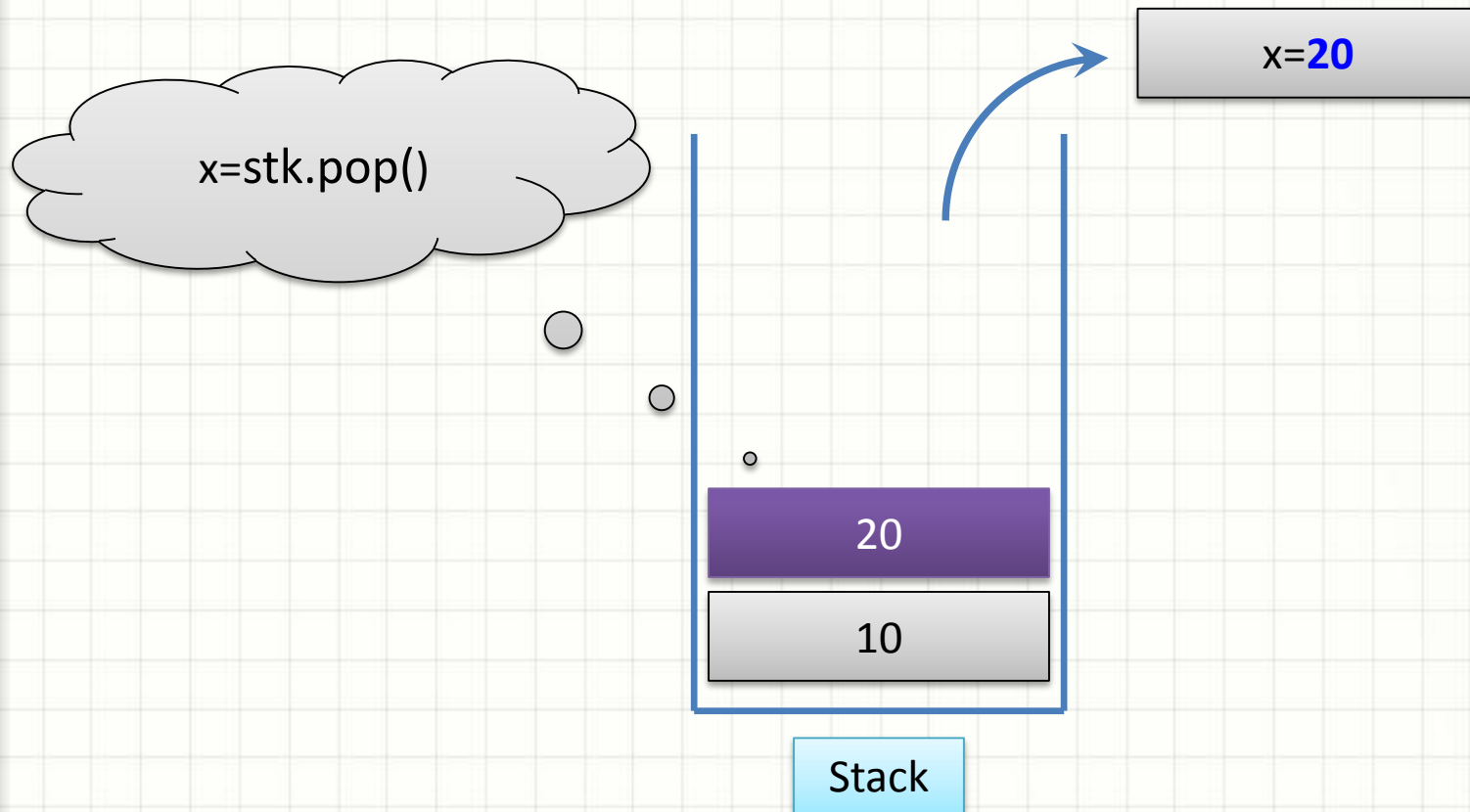
Stack Çalışma Şekli (devam...)

- Pop ile yığıtın üstünden bir eleman çıkartıyoruz.



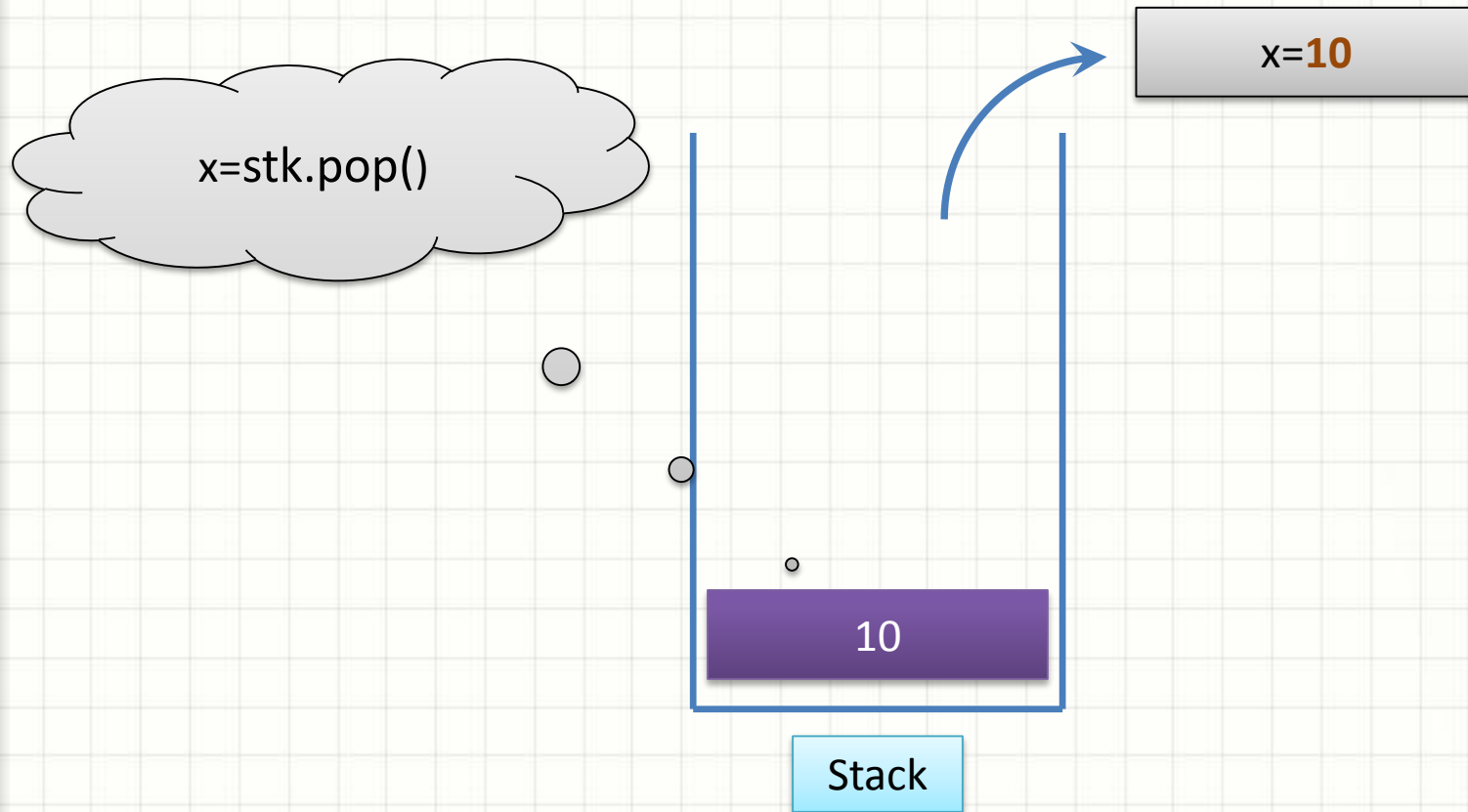
Stack Çalışma Şekli (devam...)

- Pop ile yığıtın üstünden bir eleman çıkartıyoruz.



Stack Çalışma Şekli (devam...)

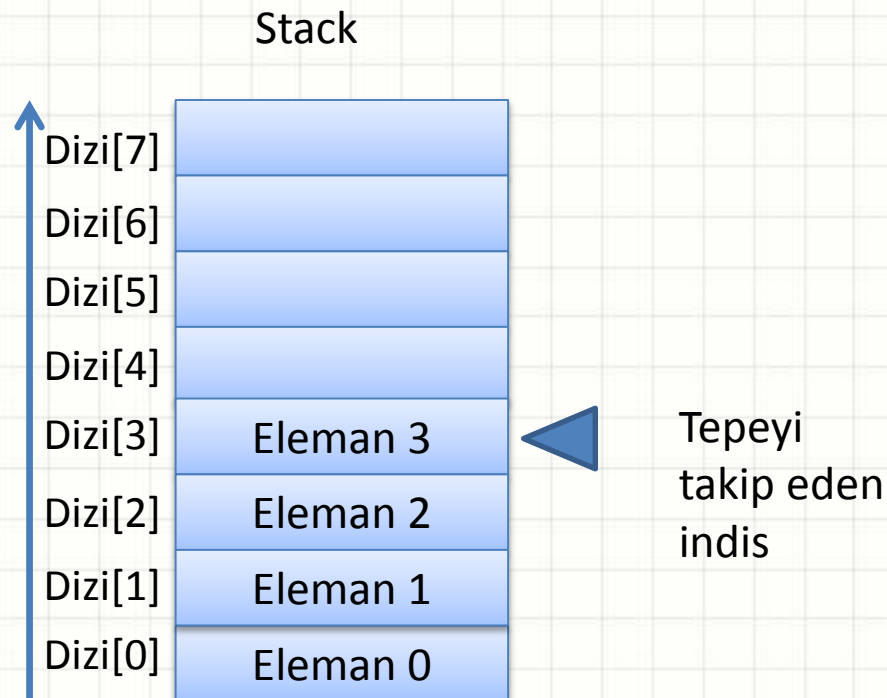
- Pop ile yığıtın üstünden bir eleman çıkartıyoruz.



Stack Implementasyonu

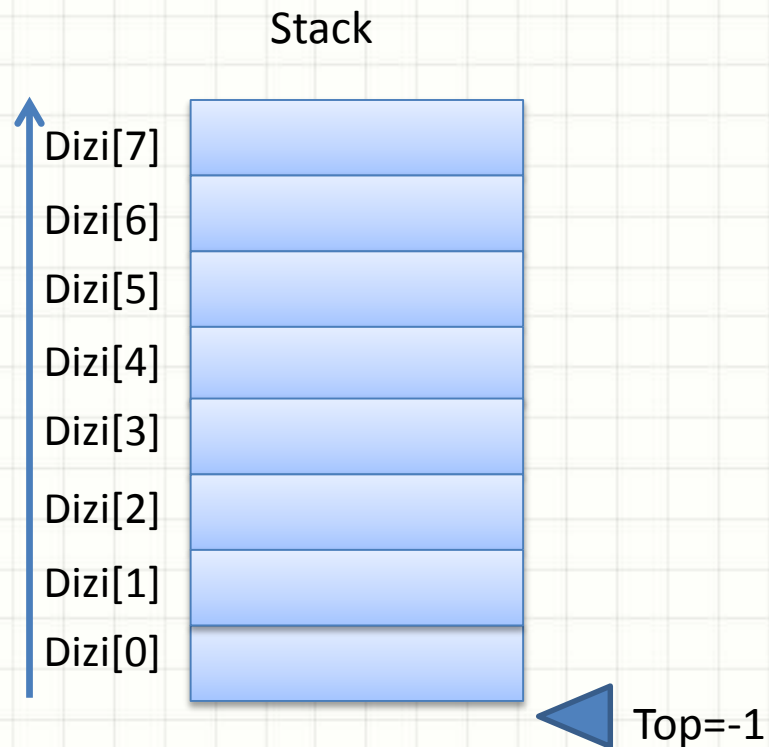
- Stack **iki şekilde** **implemente** edilebilir:
 - Dizi kullanarak
 - Bağlı liste kullanarak

Stack Dizi ile Gerçekleştirim

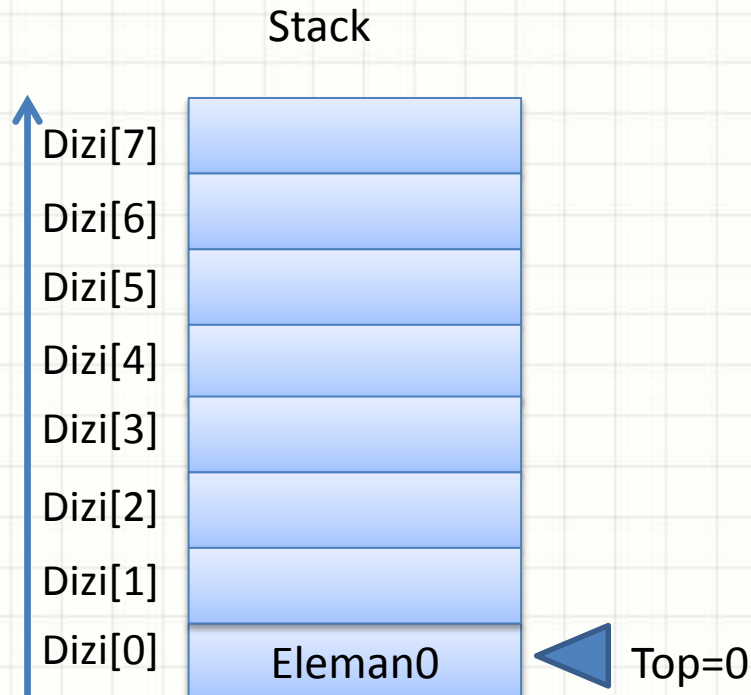


- Dizi ile gerçekleştirmede stack elemanları dizi üzerinde tutulur.
- Bir tam sayı değişken yığıtın tepesindeki elemanın adresini takip etmek için kullanılır.

Stack Dizi ile Gerçekleştirim (devam...)

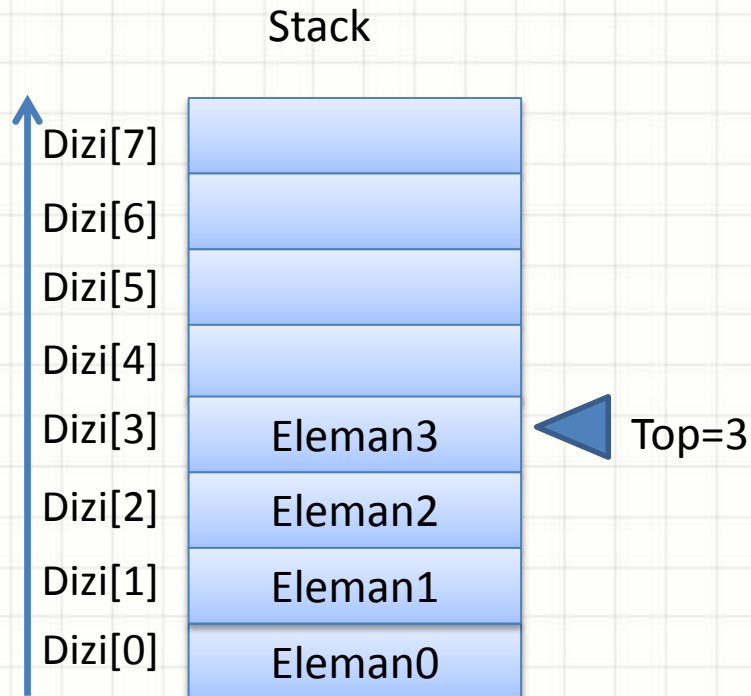


Stack Dizi ile Gerçekleştirim (devam...)



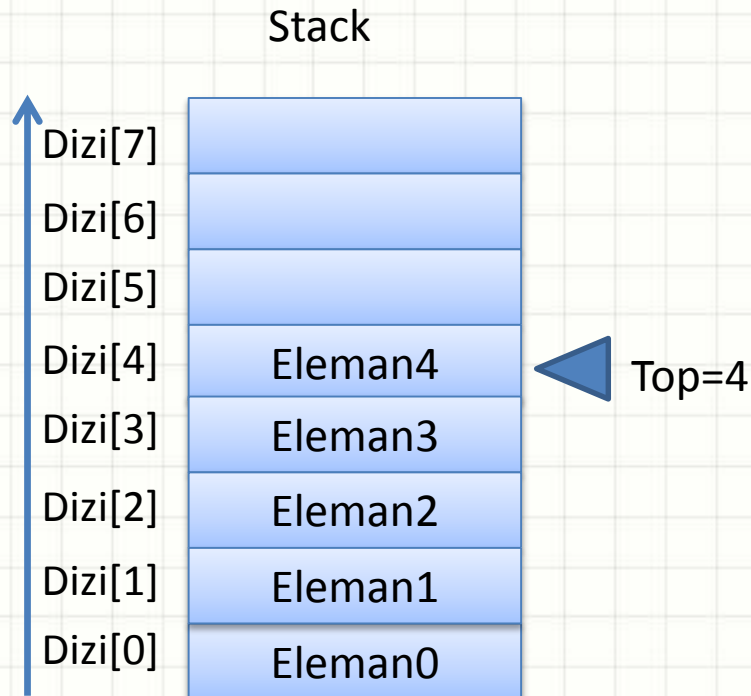
Stack.push(Eleman0)

Stack Dizi ile Gerçekleştirim (devam...)



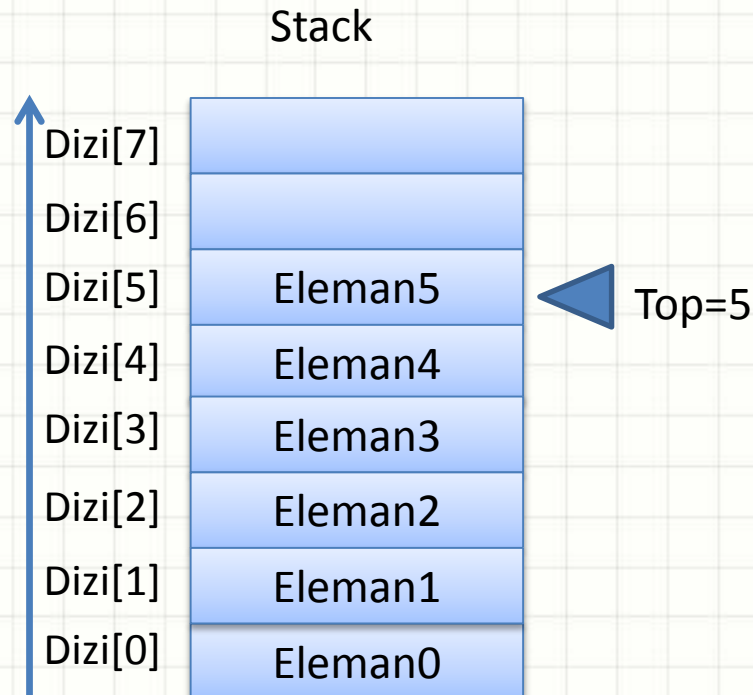
```
Stack.push(Eleman0)  
Stack.push(Eleman1)  
Stack.push(Eleman2)  
Stack.push(Eleman3)
```

Stack Dizi ile Gerçekleştirim (devam...)



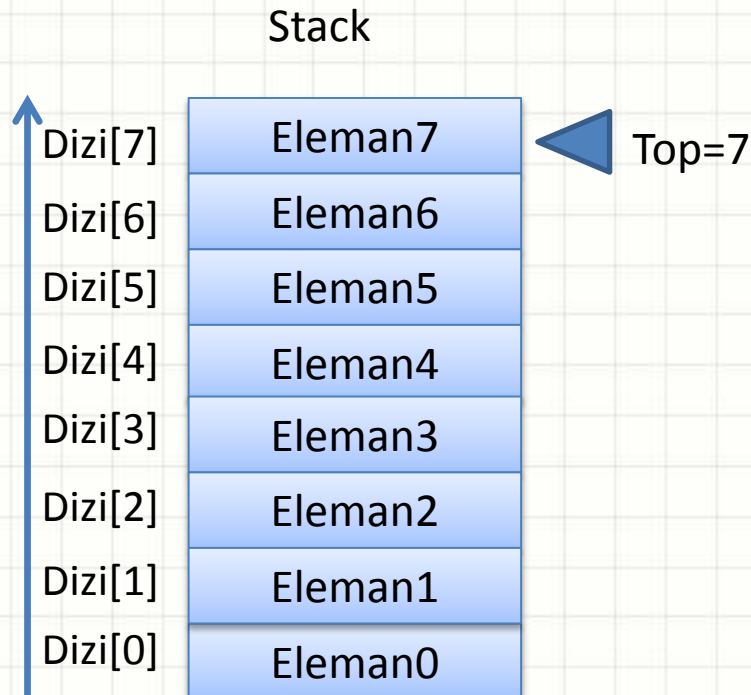
```
Stack.push(Eleman0)  
Stack.push(Eleman1)  
Stack.push(Eleman2)  
Stack.push(Eleman3)  
Stack.push(Eleman4)
```


Stack Dizi ile Gerçekleştirim (devam...)



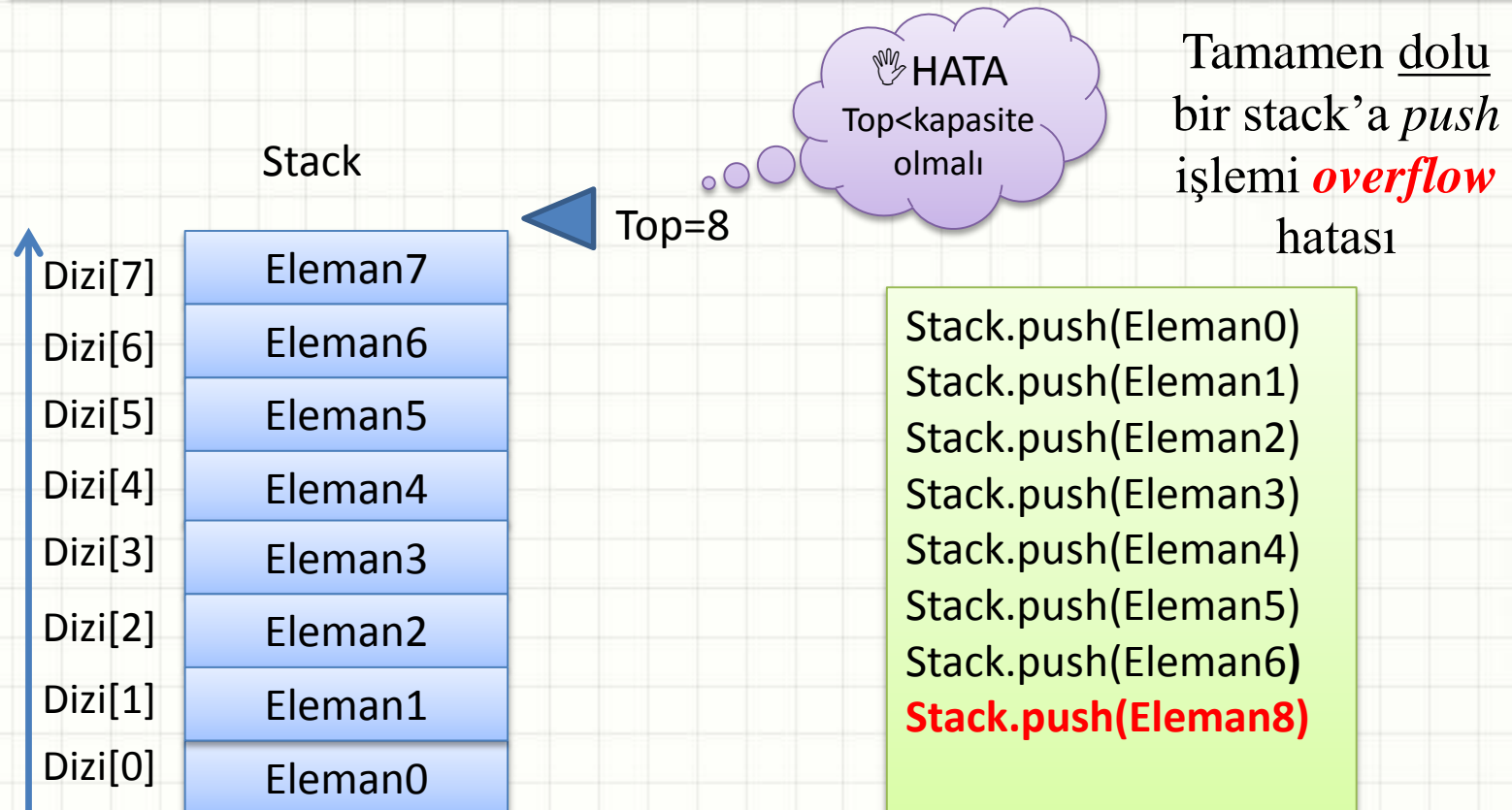
```
Stack.push(Eleman0)  
Stack.push(Eleman1)  
Stack.push(Eleman2)  
Stack.push(Eleman3)  
Stack.push(Eleman4)  
Stack.push(Eleman5)
```

Stack Dizi ile Gerçekleştirim (devam...)

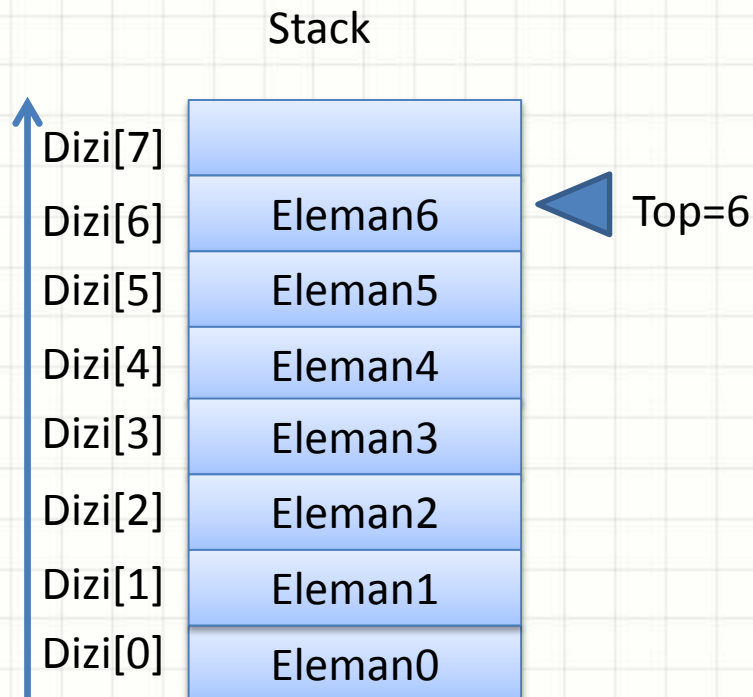


```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
```

Stack Dizi ile Gerçekleştirim (devam...)

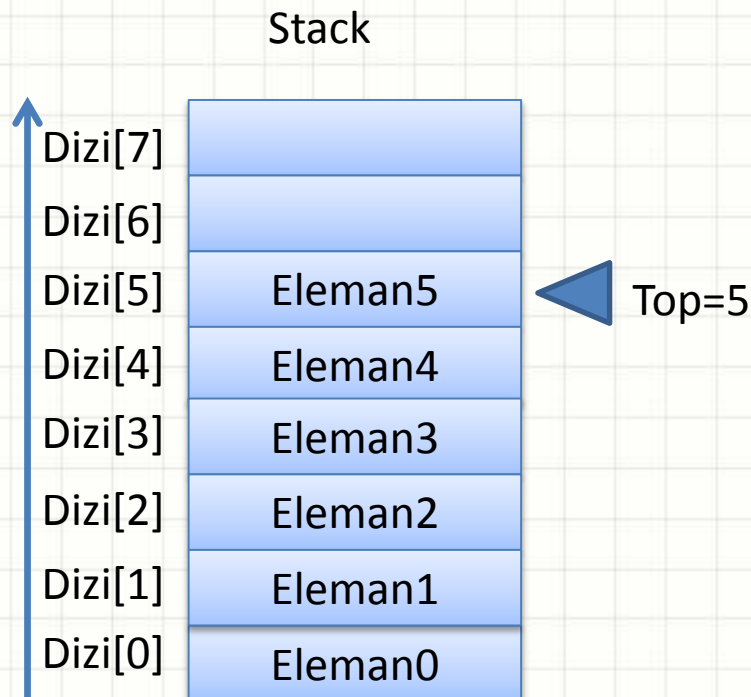


Stack Dizi ile Gerçekleştirim (devam...)



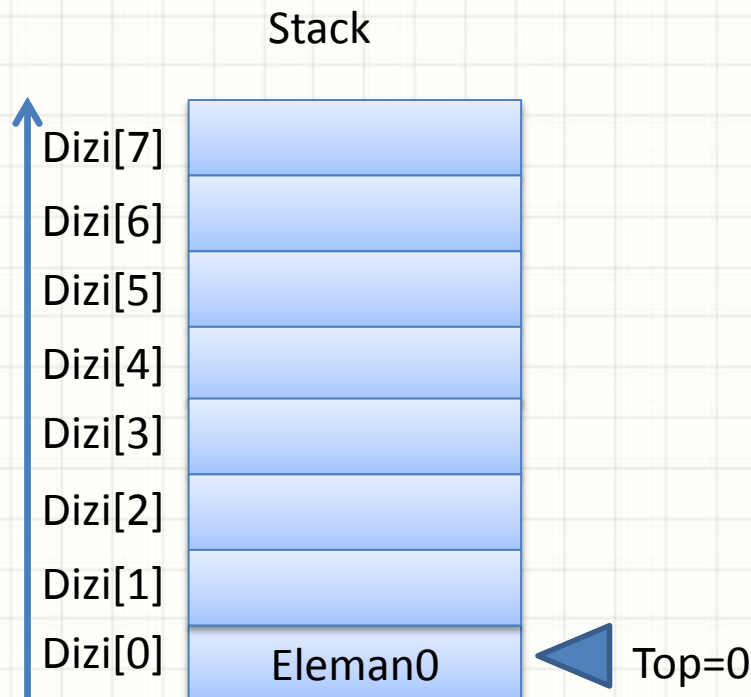
```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
Stack.pop()
```

Stack Dizi ile Gerçekleştirim (devam...)



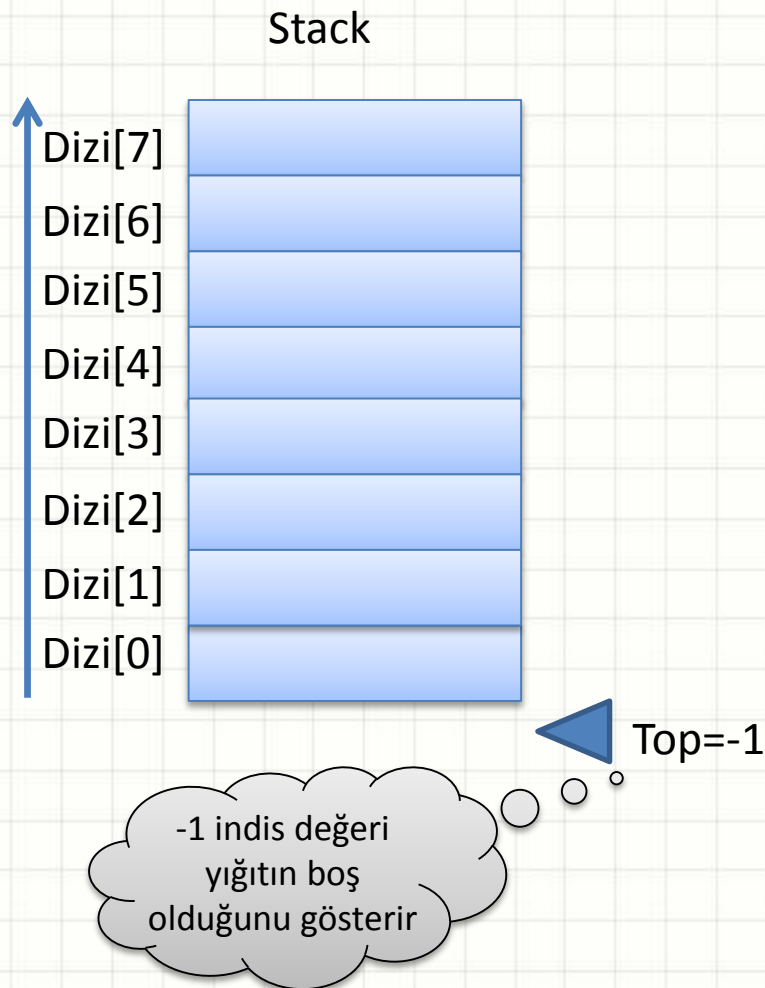
```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
Stack.pop()
Stack.pop()
```

Stack Dizi ile Gerçekleştirim (devam...)



```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
```


Stack Dizi ile Gerçekleştirim (devam...)



```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
```

Stack Dizi ile Gerçekleştirim (devam...)



Boş bir stack'ta
pop işlemi
underflow
hatası



```
Stack.push(Eleman0)
Stack.push(Eleman1)
Stack.push(Eleman2)
Stack.push(Eleman3)
Stack.push(Eleman4)
Stack.push(Eleman5)
Stack.push(Eleman6)
Stack.push(Eleman7)
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
Stack.pop()
```

Stack Uygulamaları

1. Word, Excel, Photoshop gibi yazılımlarda yapılan işlemlerin sırayla kayıt edildiği ve geri alınabilecek şekilde tutulduğu **undo fonksiyonu** bir stack uygulamasıdır.
2. Bir web tarayıcısında **ileri-geri adres gezmek** için stack yapısı kullanır.
3. C# veya Java gibi programlama dillerinde **açılan parantezin doğru kapatılması kontrolünde** (“Matching Bracket” – “Parantez Eşleştirme” kontrolü) kullanılır.
4. **Polish Notasyon:** **Infix** olarak bilinen $A*(B+C/D)-E$ cebirsel gösteriminin yerine hesap makinelerinde kullanılan **postfix** $ABCD/+*E$ notasyonuna çevirme işleminde stack kullanır.
5. HTML-XML’de **tag’lerin eşleştirilmesi** bir stack uygulamasıdır.

Stack Uygulamaları (devam...)

5. Stack'ların bir diğer uygulama alanı **labirent** türü problemlerin çözümünde **backtracking** (bir yola gir yol tıkanırsa en son yol ayırımına geri gel, başka yola devam et!) yöntemiyle kullanılır.
 - **Yol bilgisi** bir stack yapısına **push** edilir **yol yanlışsa** son gidilen yanlış nokta **pop edilir** önceki noktayla devam edilir.
6. Java derleyicisi program kodunun tamamını *postfix'e* çevirirken stack kullanır.
7. Java Virtual Machine (JVM) **byte code'ları** **execute ederken** altyapısında yine stack kullanır.
8. **Recursion** ve **function call** işlemlerinin Bellekte gerçekleştirilmesinde stack kullanılır.

Uyg1-String Reverse

- Amacımız string bir ifadeyi tersten yazdırmaktır.

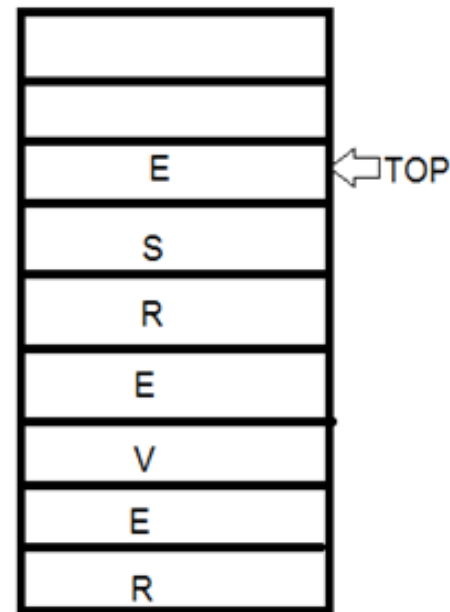
REVERSE → ESREVER

- Çözüm?:
 - String ifadedeki her bir karakter soldan-sağa okunarak, stack'e **Push** metodu ile eklenir.
 - Stack'deki her bir karakter **Pop**'ile stack'den geriye doğru okunur ve silinir.

Uyg1-String Reverse (devam...)

STRING IS:

REVERSE



STACK

Uyg2-Parantez Eşleştirme

- Matematiksel bir ifadede veya program kodundaki parantez eşleştirme kontrolünü yapmak için stack kullanılabilir.

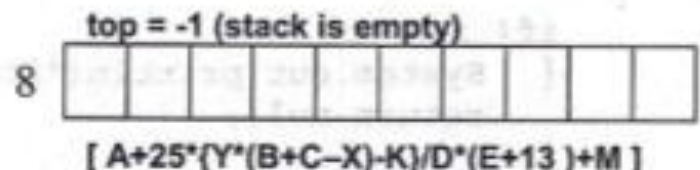
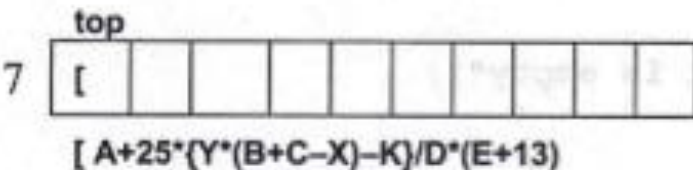
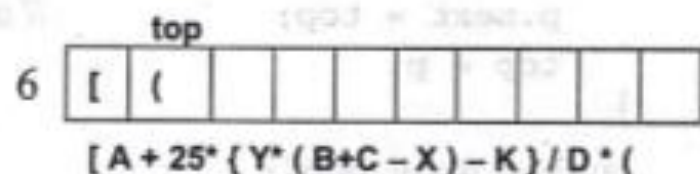
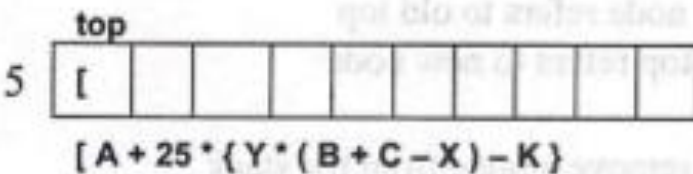
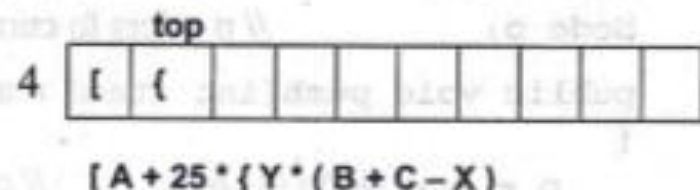
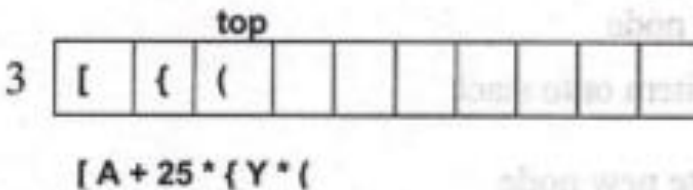
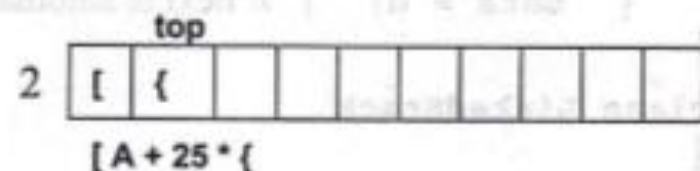
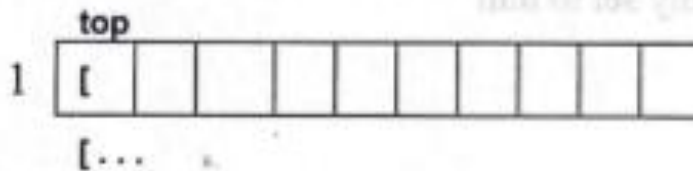
Geçerli İfadeler	Geçersiz İfadeler
{ }	{ (}
({ [] })	([(()])
{ [] () }	{ } [])
[{ ({ } [] ({ }))]	[{ }) { (] }]

Uyg2-Parantez Eşleştirme (devam...)

- Algoritma? $[A+25*\{Y*(B+C-X)-K\}/D*(E+13)+M]$
 - Bir açılış parantezi ile **karşılaşıldığında** “(”, “{”, “[” stack’e **Push** edilir.
 - Bir kapanış parantezi ile **karşılaşıldığında** “)”, “}”, “]” stack’e bakılır, **stack boş değilse** stack’ten bir eleman **Pop** edilerek, *doğru karşılık olup olmadığı kontrol edilir.*
 - ❑ **Doğruysa** işlem sürdürülür.
 - ❑ **Doğru değilse** ifade geçersizdir.
 - Stack sonuna ulaşıldığında *stack boş olmalıdır.*
 - ❑ Aksi halde **açılmış** ama **kapanmamış** parantez olabilir.

Uyg2-Parantez Eşleştirme (devam...)

Test: $[A+25*\{Y*(B+C-X)-K\}/D*(E+13)+M]$



Polish Notasyon

- Polish notasyonu **Bilgisayar Bilimleri** alanındaki **önemli konulardan bir tanesidir**. Operatörleri, operandlardan önce veya sonra gösterme metodu olarak tanımlanabilir.
 - Infix: Bilinen klasik gösterim.
 - Prefix: Operatörler operandlardan önce yazılır.
 - Postfix: Operatörler operandlardan sonra yazılır.

Polish Notasyon (devam...)

Örnek: $A+B$

- Operatör (işlemci) : +
- Operand (işlenenler) A, B
- Infix: $A+B$
- Prefix: $+AB$ (benzer bir gösterim **add(A,B)** fonksiyonu)
- Postfix: $AB+$

Infix	Postfix
$A+B-C$	$AB+C-$
$(A+B)*(C-D)$	$AB+CD-*$
$A^B*C-D+E/F/(G+H)$	$AB^C*D-EF/GH+/+$

Polish Notasyon (devam...)

- **Postfix** formda parantez kullanımına **gerek yoktur**.
- Infix \rightarrow Postfix forma çevrilen bir ifadede operand'ların bağlı olduğu operator'leri (+,-,*,/) görmek zorlaşır
 - 3 4 5 * + ifadesinin sonucunun **23**'e,
 - 3 4 + 5 * ifadesinin sonucunun **35**'e karşılık geldiğini bulmak
 - Infix gösterime alışık olduğumuz için zor gibi görünür.
- Fakat *parantez kullanmadan* **tek anlama gelen hale dönüşür**. İşlemleri, **hesaplamaları yapmak kolaylaşır**.
- **Birçok derleyici** $3*2+5*6$ gibi bir Infix ifadenin değerini hesaplayacağı zaman Postfix forma dönüştürdükten (belirsizliği ortadan kaldırdıktan sonra) sonucu hesaplar : “3 2 * 5 6 * +”
- Hem **Infix \rightarrow Postfix dönüşümünde** hem de **Postfix hesaplamasında** **stack** kullanılabilir.

Uyg4-InfixToPostfix

- $(10+20)*(30+40)/(50+60)$
- Stack :
- Çıkış :

Uyg4-InfixToPostfix (devam...)

- **(10+20)*(30+40)/(50+60)**

- **Stack : (**

- **Çıkış :**


Uyg4-InfixToPostfix (devam...)

- $(\mathbf{10}+20)*(30+40)/(50+60)$

- Stack : (

- Çıkış : 10

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50+60)$

- Stack : (+
- Çıkış : 10

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50+60)$
- Stack : (+
- Çıkış : 10 20


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50+60)$


- Stack : ~~(~~+

- Çıkış : 10 20 +


Uyg4-InfixToPostfix (devam...)

- $(10+20)^*(30+40)/(50+60)$
- Stack :  $\times ($
- Çıkış : 10 20 +

Uyg4-InfixToPostfix (devam...)

- $(10+20) * (30+40) / (50+60)$

- Stack : *
- Çıkış : 10 20 +


Uyg4-InfixToPostfix (devam...)

- $(10+20)* (30+40)/(50+60)$

- Stack : * (
- Çıkış : 10 20 +


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(\textcolor{red}{30}+40)/(50-60)$
- Stack : * (
- Çıkış : 10 20 + $\textcolor{red}{30}$

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : * (+ 
- Çıkış : 10 20 + 30

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : * (+ 
- Çıkış : 10 20 + 30 40


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$

- Stack : * ~~(+)~~

- Çıkış : 10 20 + 30 40 +


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : 
- Çıkış : 10 20 + 30 40 +


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : ~~*~~*
- Çıkış : 10 20 + 30 40 + *

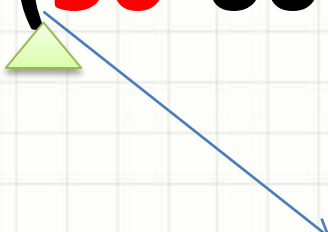
Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : 
- Çıkış : 10 20 + 30 40 + *


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : $/ ($ 
- Çıkış : 10 20 + 30 40 + *

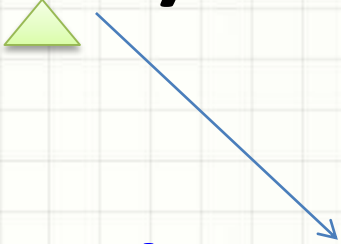
Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$ 
- Stack : / (
- Çıkış : 10 20 + 30 40 + * 50

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : $/ (-$ 
- Çıkış : 10 20 + 30 40 + * 50

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$

- Stack : / (-
- Çıkış : 10 20 + 30 40 + * 50 60


Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$

- Stack : ~~/~~-

- Çıkış : 10 20 + 30 40 + * 50 60 -

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : ~~*)~~ 
- Çıkış : 10 20 + 30 40 + * 50 60 -

Uyg4-InfixToPostfix (devam...)

- $(10+20)*(30+40)/(50-60)$
- Stack : /
- Çıkış : 10 20 + 30 40 + * 50 60 - /

InfixToPostfix Algoritma

- Sol parantez ise: Sol parantez yığına **Push** edilir.
- Sağ parantez ise: Sol parantez çıkana kadar yığından **Pop** işlemi yapılır. Alınan işlem işareti **Postfix** ifadeye eklenir. Sol parantez görüldüğünde **Pop** işlemine son verilir. Sol parantez **Postfix**'e eklenmez.
- Sayı ise: **Postfix** ifadeye eklenir.
- İşlem işareti ise: Yığının en üstünde sol parantez varsa veya en üstteki işaretin önceliği bu işaretten düşük ise işlem işareti yığına **Push** edilir. Bu işaretin önceliği daha düşük ise yığındaki bu işaretten yüksek öncelikli işaretler için **Pop** işlemi yapılır. Stackten **Pop** edilenler **Postfix** ifadeye eklenir. İşlem işareti yığına **push** edilir.
- İfadeler bittiğinde: Yığındaki işaretler sıra ile **Pop** edilerek postfix ifadeye eklenir.

Postfix Çözümleme Algoritma

- Postfix ifade soldan sağa doğru değerlendirilir. Eğer o anda bakılan:
 - Sayı ise: Sayı yığına **push** edilir.
 - İşlem işareti ise:
 - Yığının üstündeki iki değer **pop** edilerek aralarında bu işlem yapılır.
 - İşlem sonucu yığının en üstüne **push** edilir.

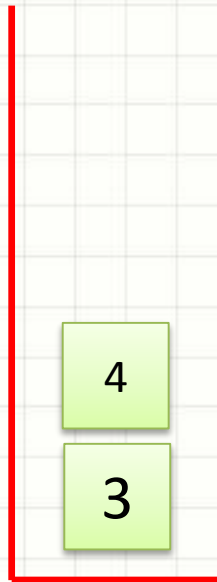
Uyg5-Postfix Değerlendirme

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



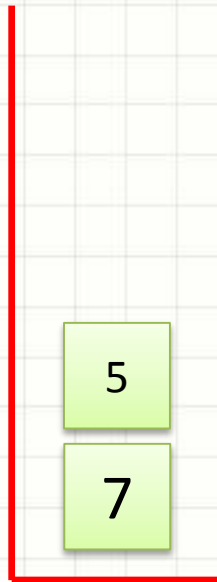
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



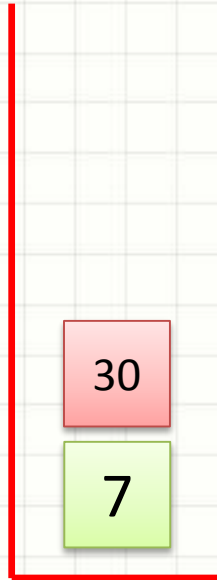
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



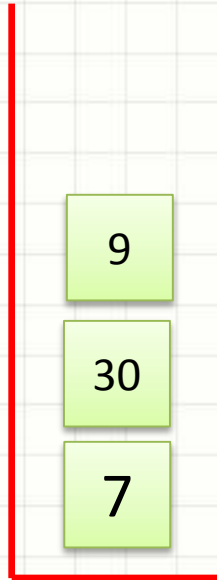
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



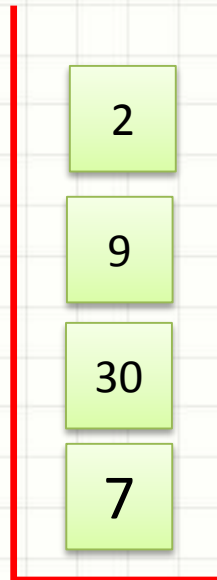
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



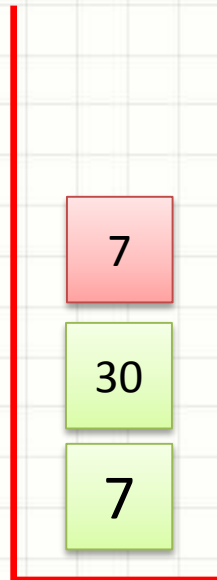
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



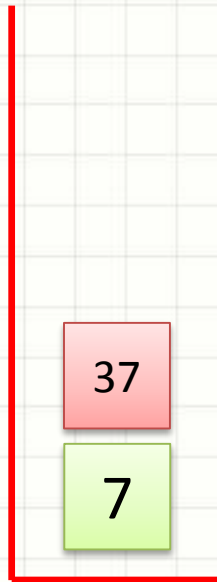
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



259

Stack İşlem Karmaşıklığı

- Dizi ile tanımlanan stack'ta boyut önceden belirli olmalıdır.
- Bu bağlamda stack'taki eleman sayısı n olarak kabul edilirse bir stack için (**dizi** ve **bağlı liste** için) işlem karmaşıklıkları aşağıdaki gibidir.

Space Complexity (for n push operations)	$O(n)$
Time Complexity of push()	$O(1)$
Time Complexity of pop()	$O(1)$
Time Complexity of size()	$O(1)$
Time Complexity of isEmpty()	$O(1)$
Time Complexity of isFullStack()	$O(1)$
Time Complexity of deleteStack()	$O(1)$