

BMT207

Veri Yapıları

Dr.Günay TEMÜR
Düzce Üniversitesi
Bilgisayar Mühendisliği

BÖLÜM - 4

İçerik;

- Stack (Yığın) Veri Yapısı Modeli
- Stack Çalışma Şekli
- Stack Dizi Implementasyon
- Liste Implementasyon
- Stack Uygulamaları

Stack ?

- Stack, doğrusal artan bir veri yapısı modeli olup;
- Insert (**push**) ve Delete (**pop**) işlemleri,
 - Listenin sadece **“top”** adı verilen hafıza alanında gerçekleştirilir
 - **“top”**: stack’in en üst hafıza alanıdır.
- Bu nedenle stack;
 - **Son Giren İlk Çıkar**
 - **(Last In First Out - LIFO)** mantığı ile çalışan bir veri yapısı modelidir.

Stack ?

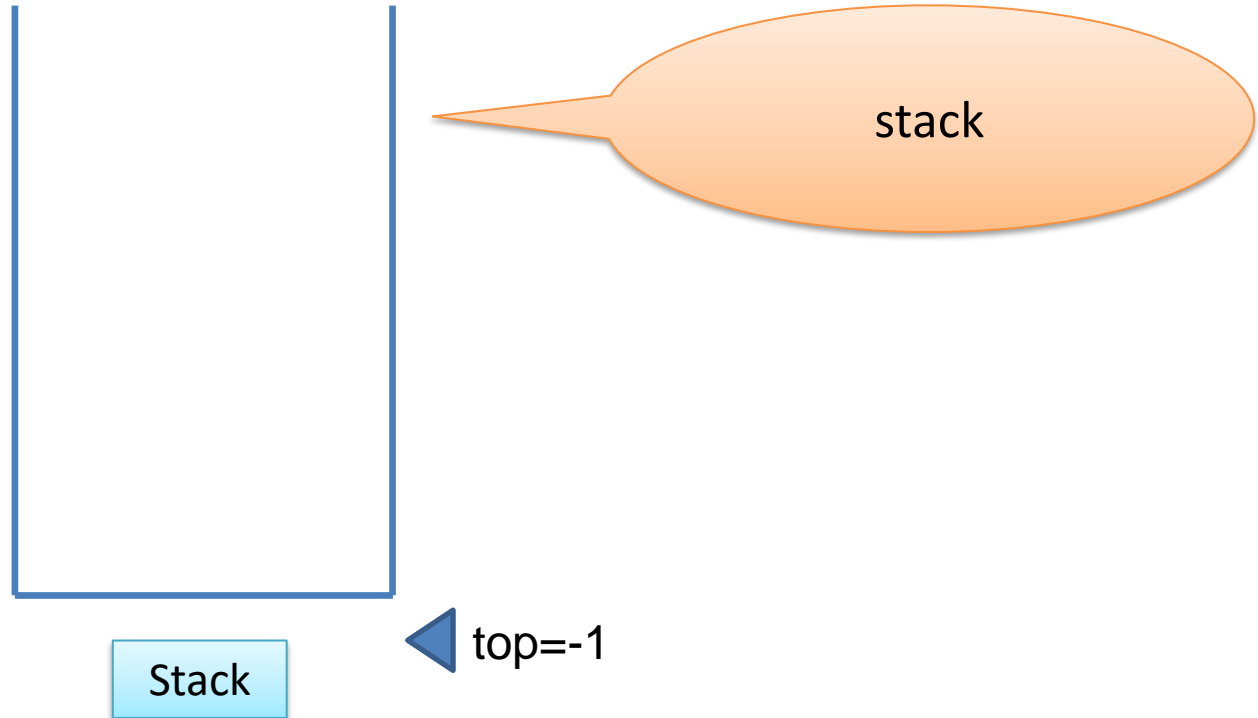


İlk Hangi Elemanı
Alırsınız



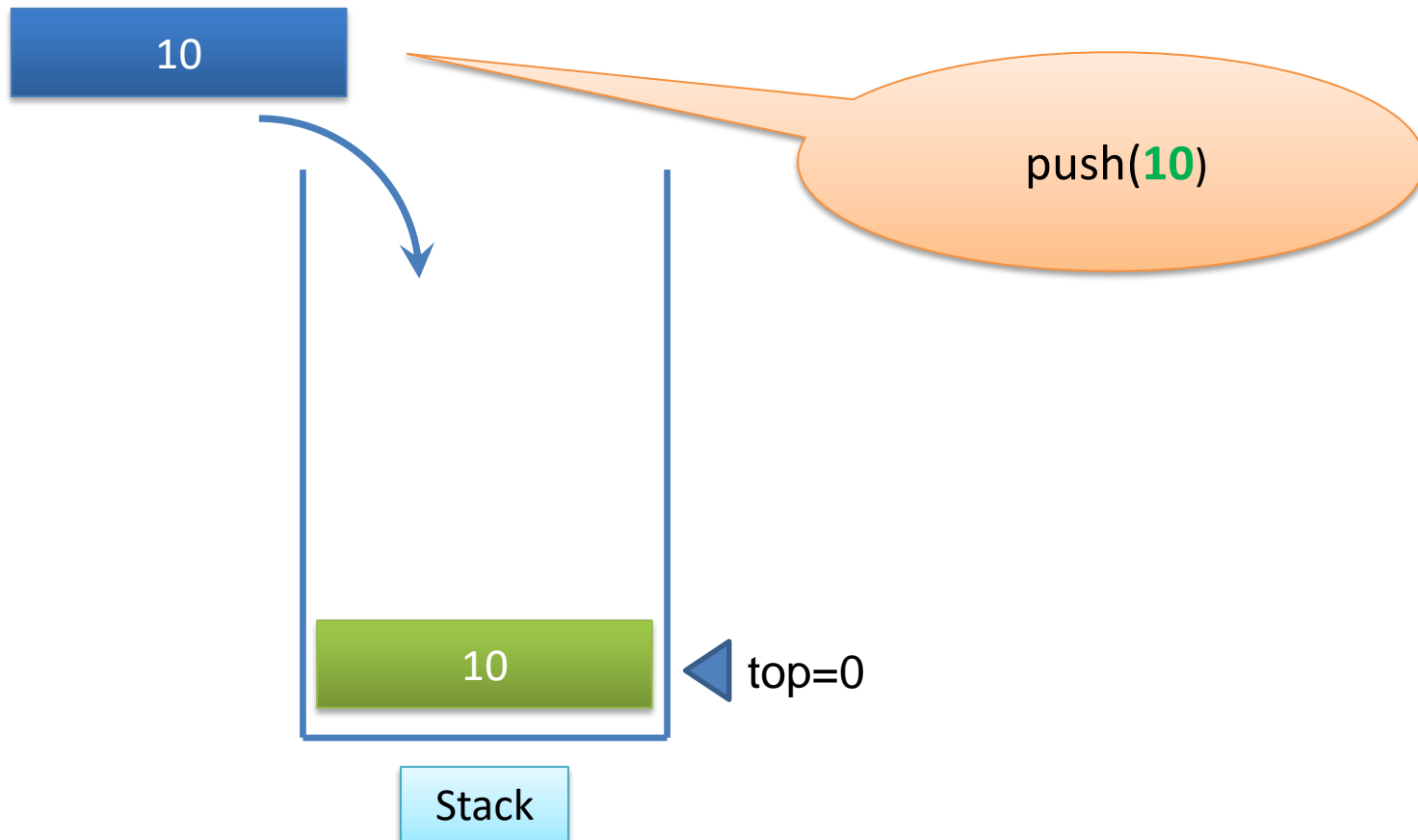
Stack Çalışma Mantığı

- Stack olarak tanımlanmış bir hafıza modeliniz olsun.
- İlk aşamada **stack hafıza boş**.



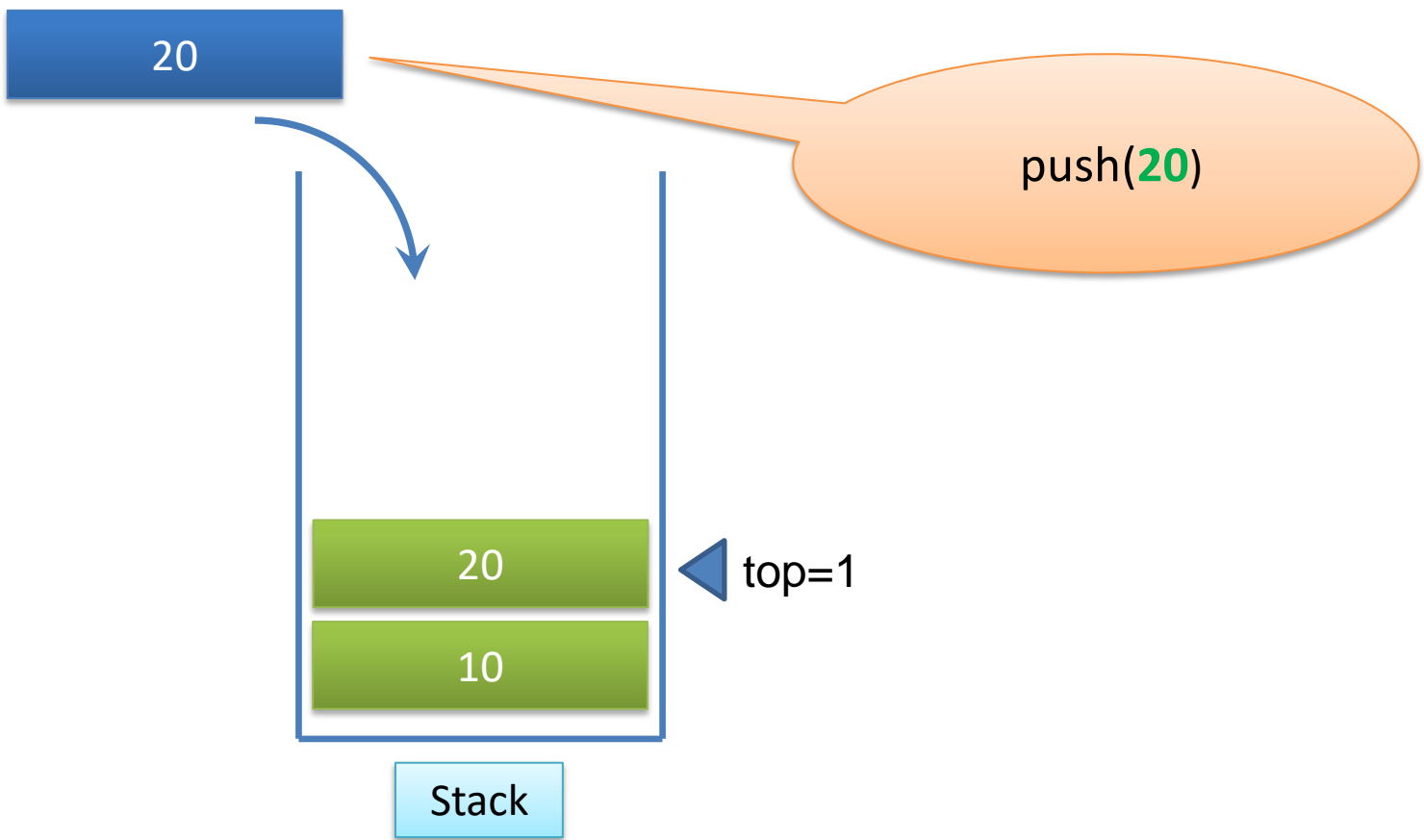
Stack Çalışma Mantığı (devam...)

- **push() fonksiyonu**, yığıtın üstüne yeni bir eleman ekler.



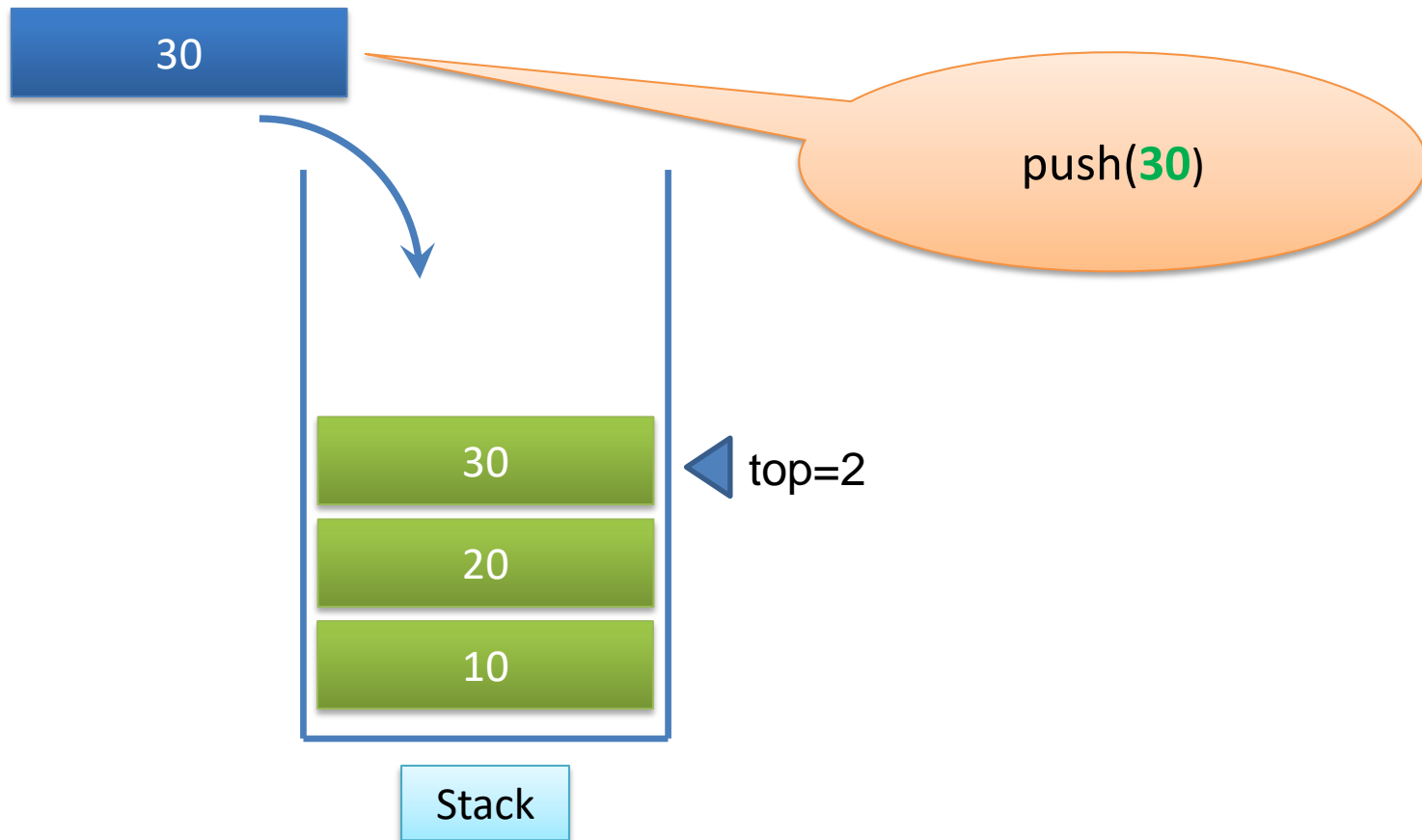
Stack Çalışma Mantığı (devam...)

- yeni bir eleman daha...



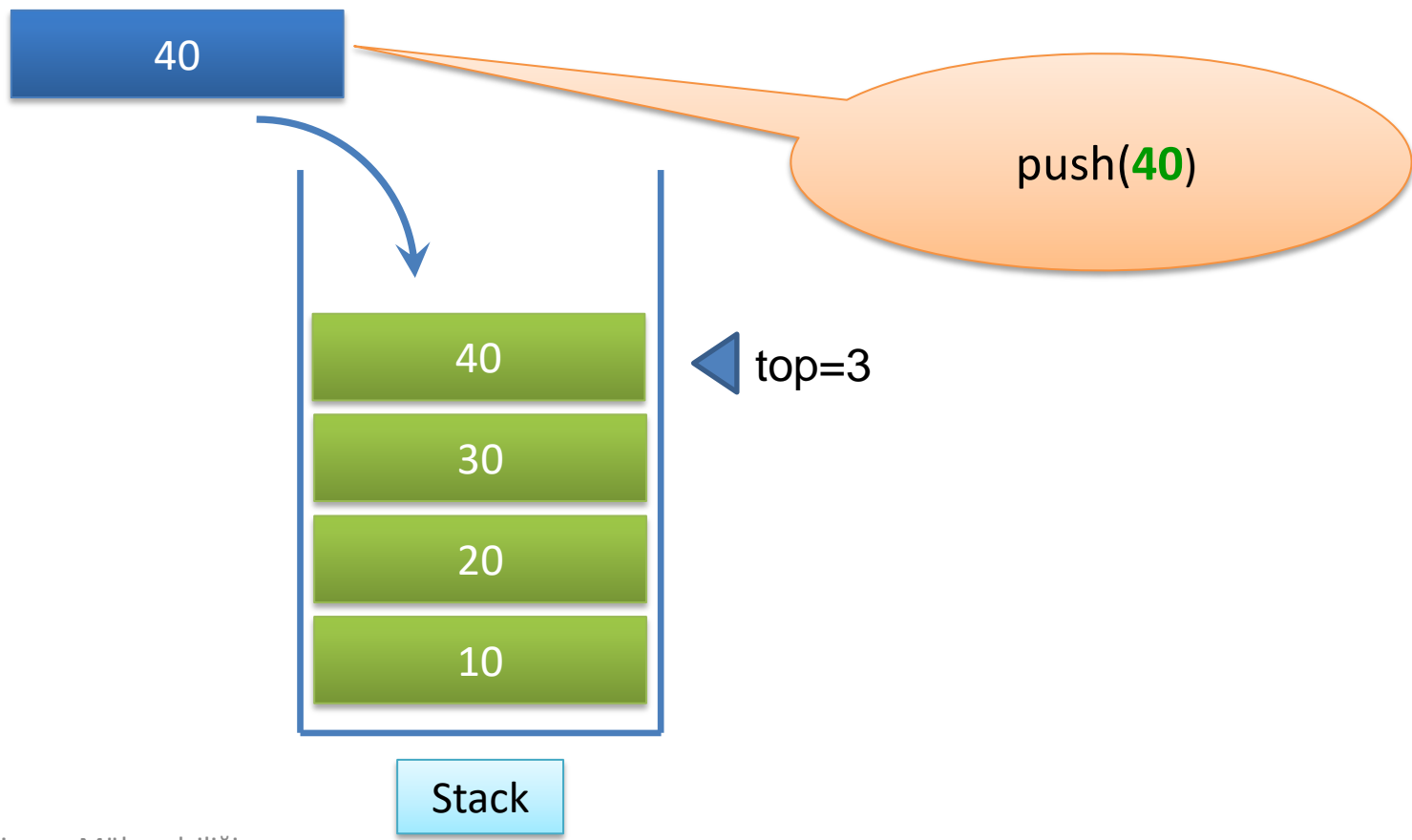
Stack Çalışma Mantığı (devam...)

- yeni bir eleman daha...



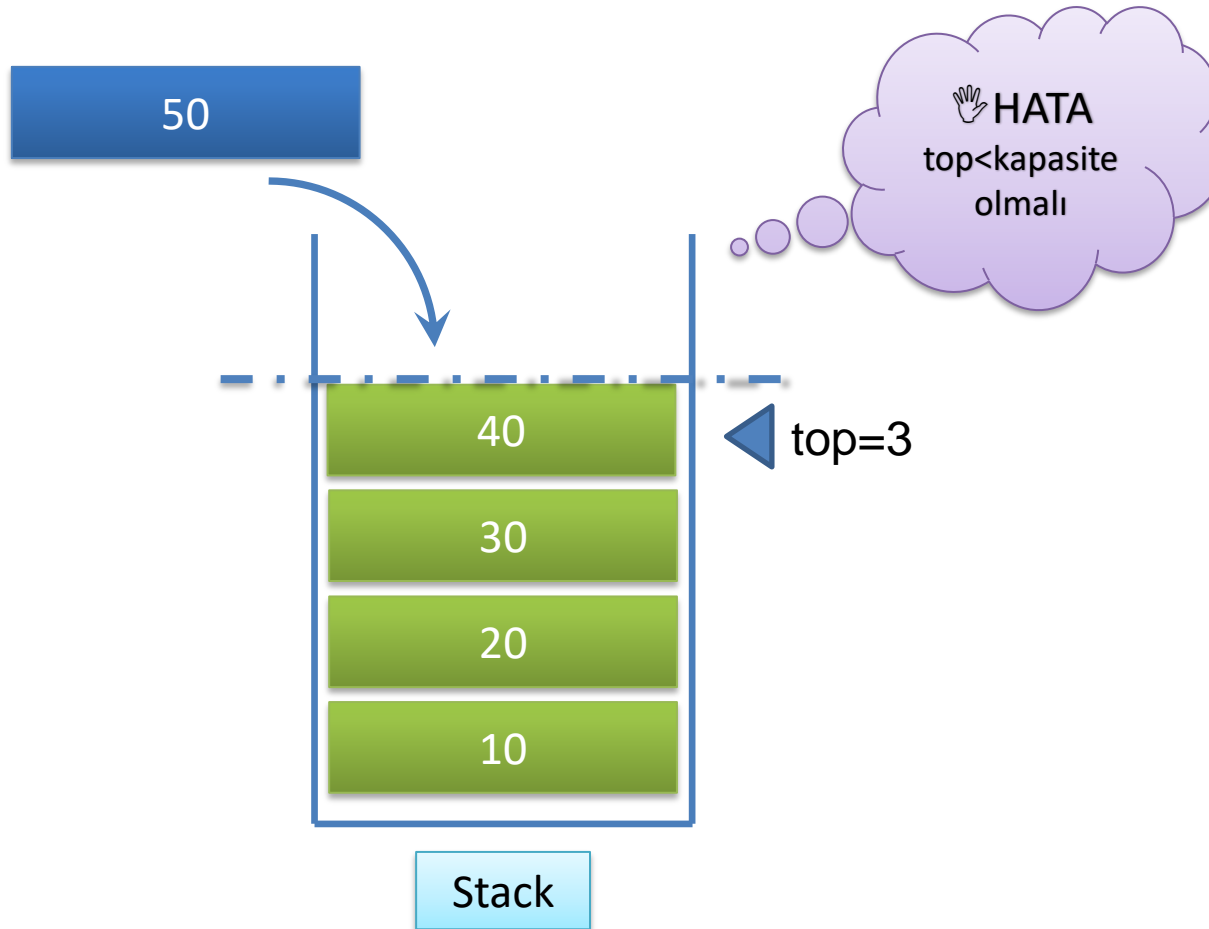
Stack Çalışma Mantığı (devam...)

- yeni bir eleman daha...



Stack Çalışma Mantığı (devam...)

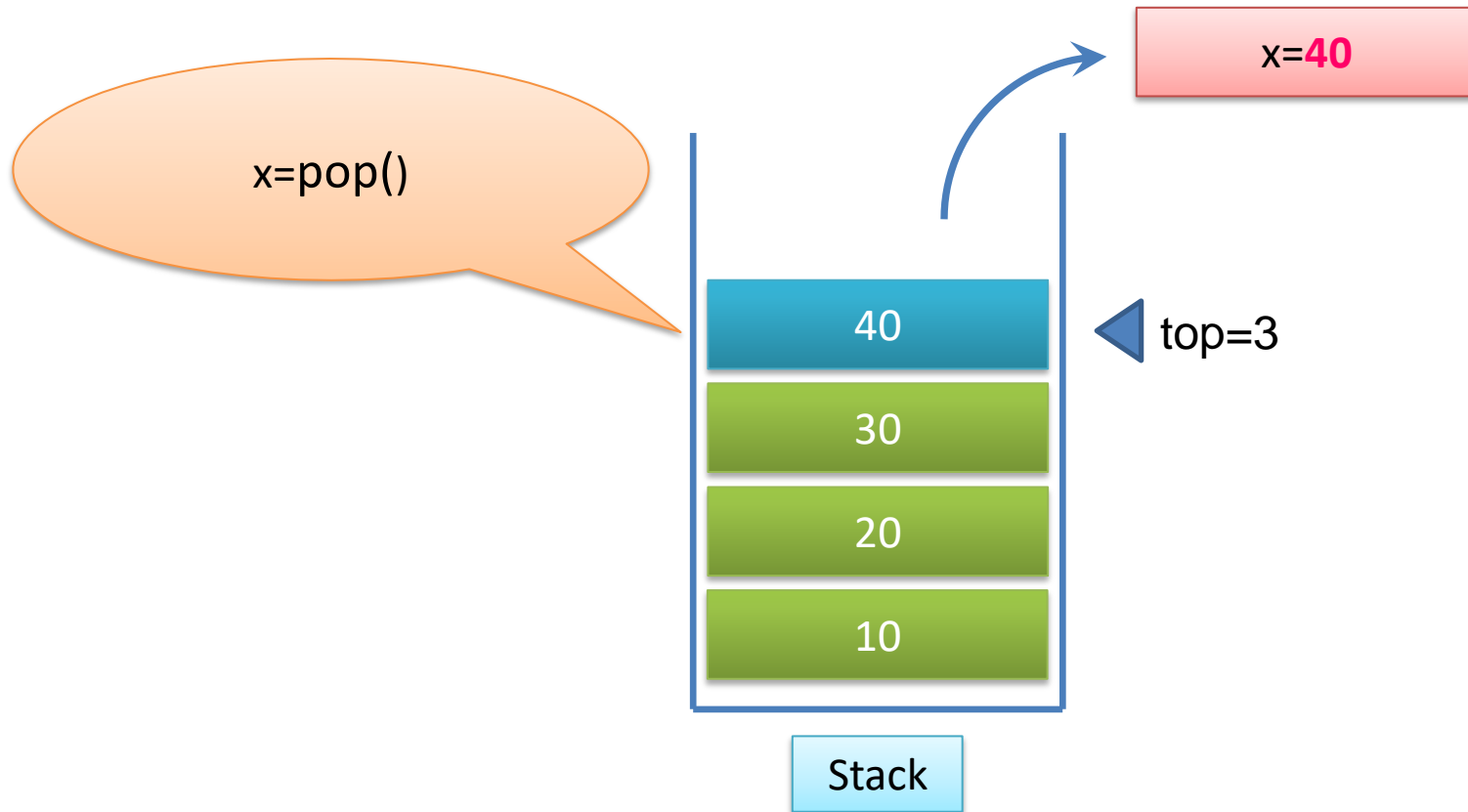
- yeni bir eleman daha...



Tamamen dolu
bir stack'a *push*
işlemi **overflow**
hatası

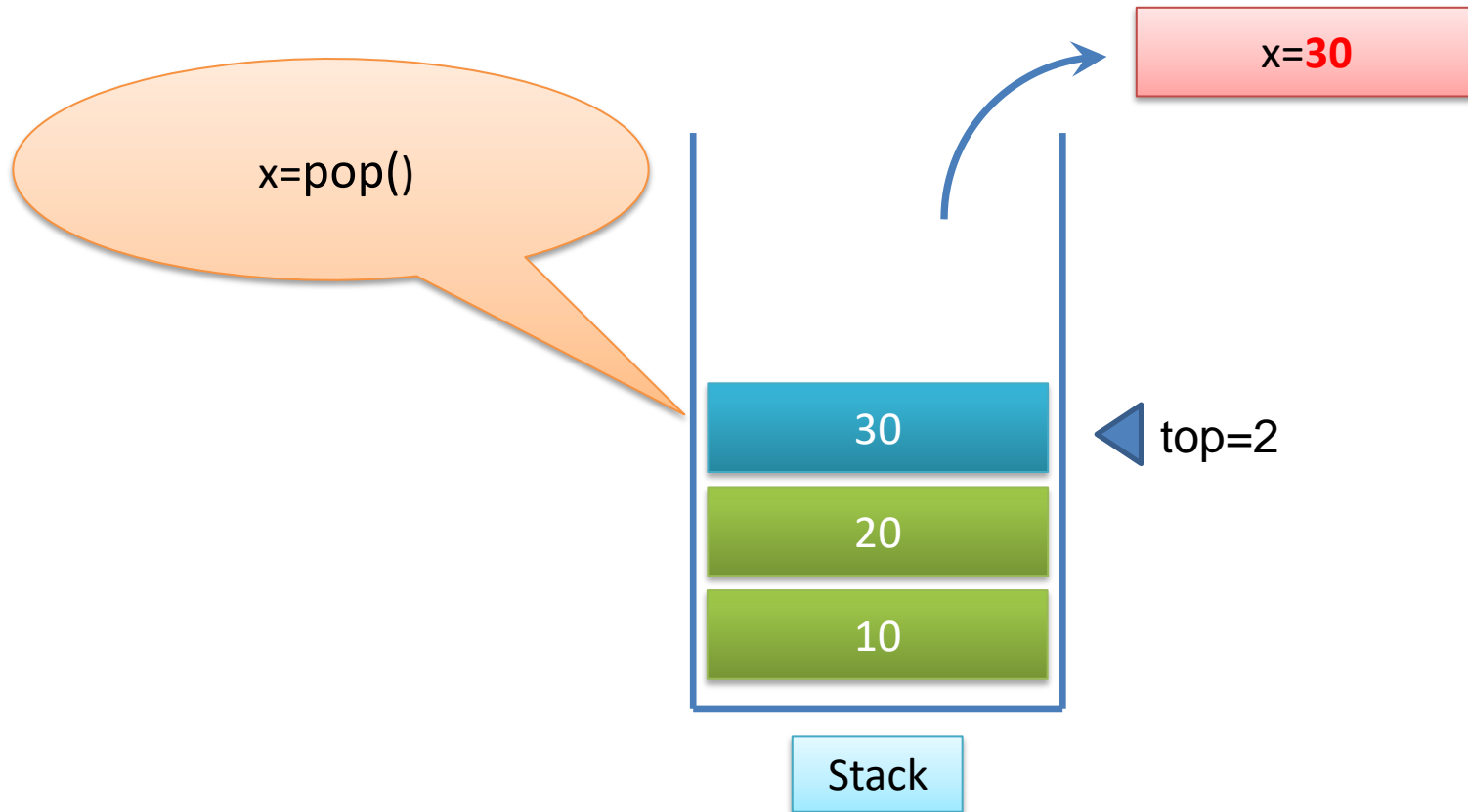
Stack Çalışma Mantığı (devam...)

- **pop()** fonksiyonu, stack hafızadan bir eleman çıkartır.



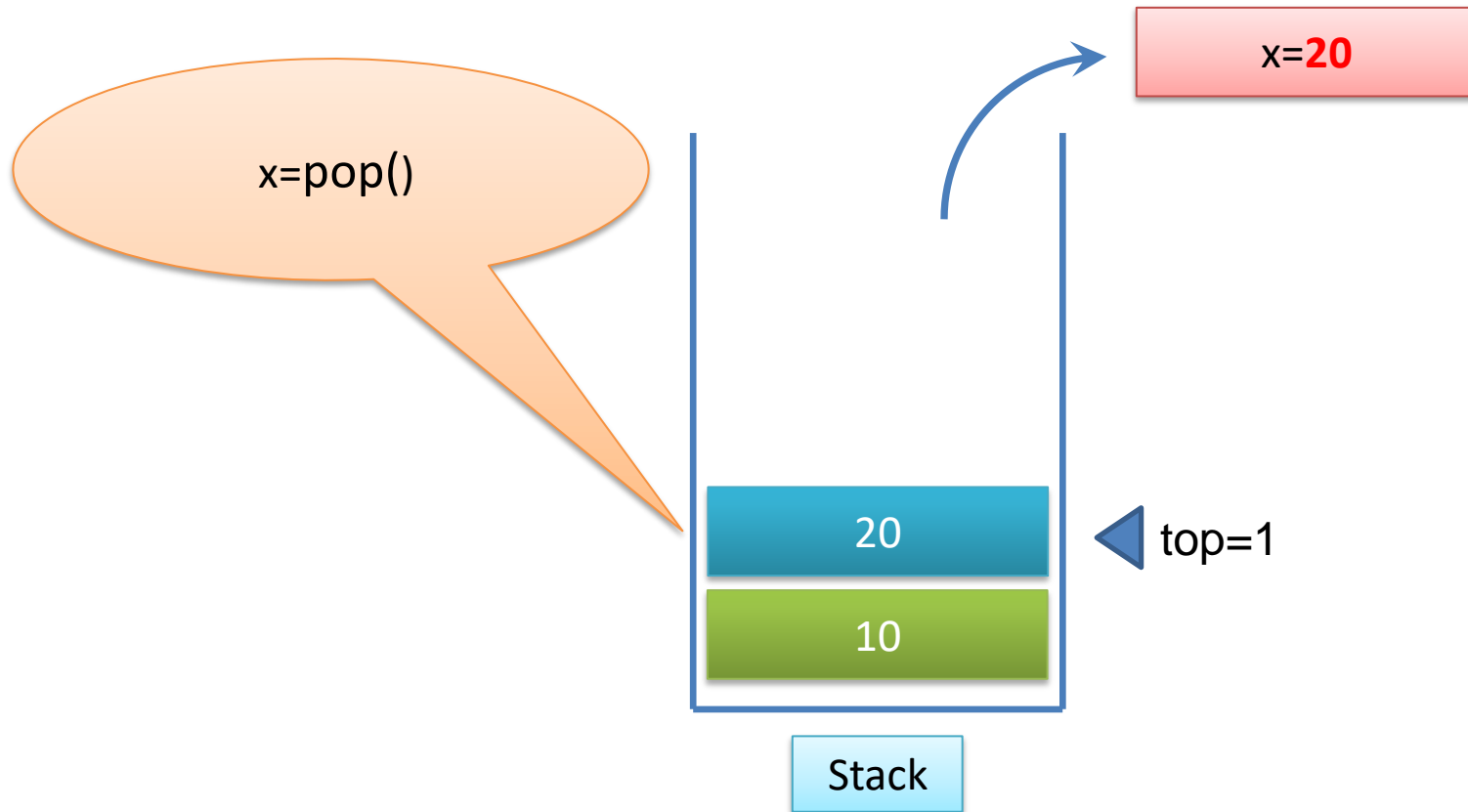
Stack Çalışma Mantığı (devam...)

- bir eleman daha...



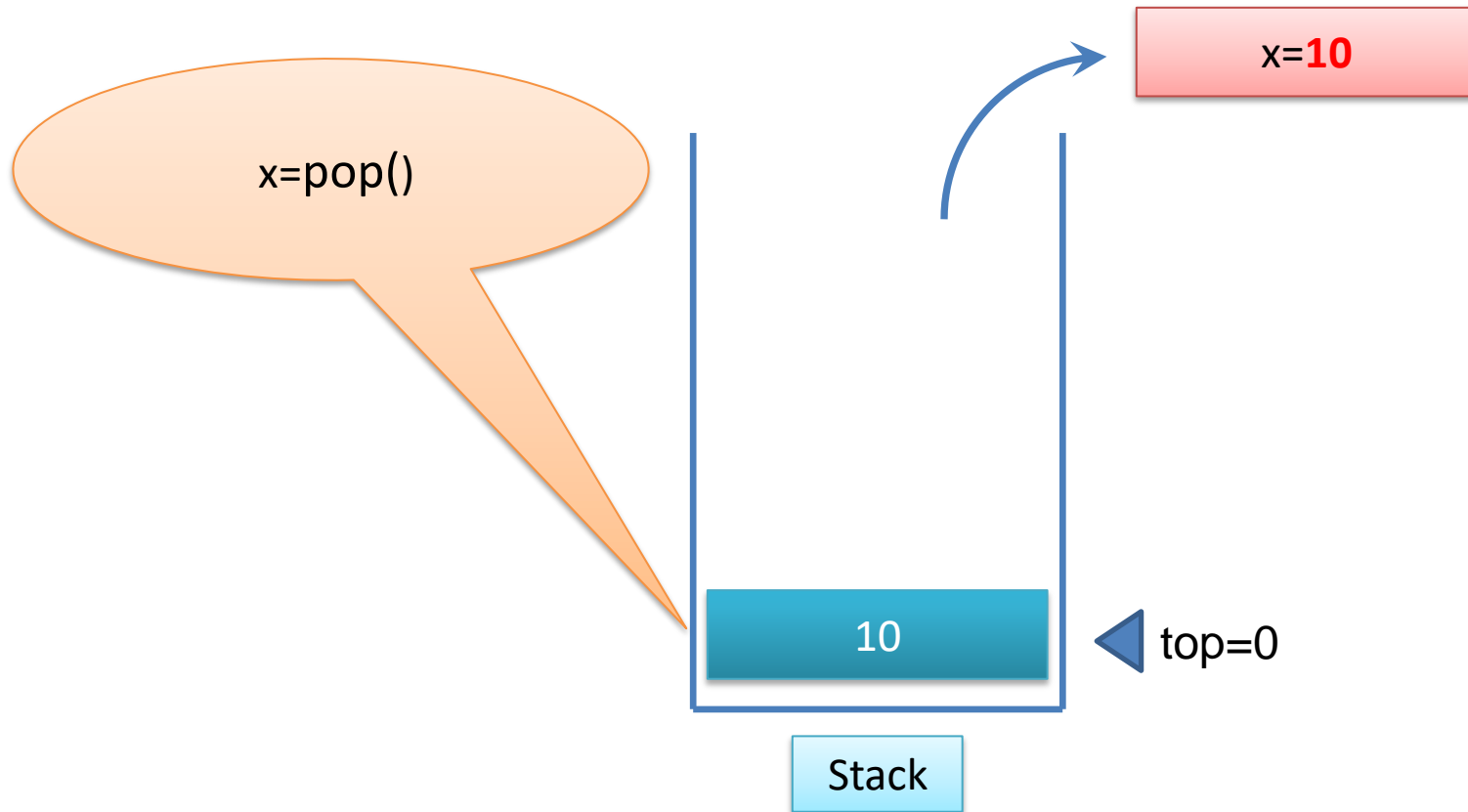
Stack Çalışma Mantığı (devam...)

- bir eleman daha...



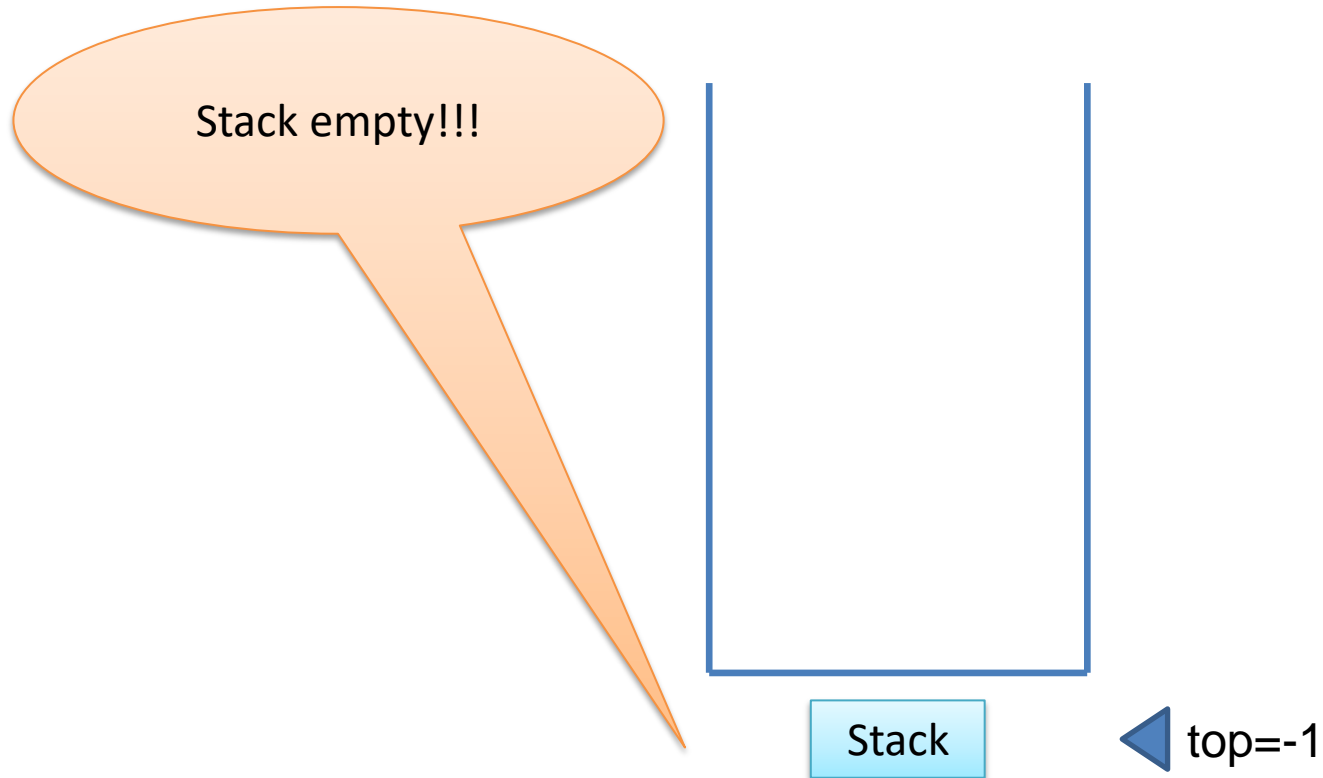
Stack Çalışma Mantığı (devam...)

- bir eleman daha...



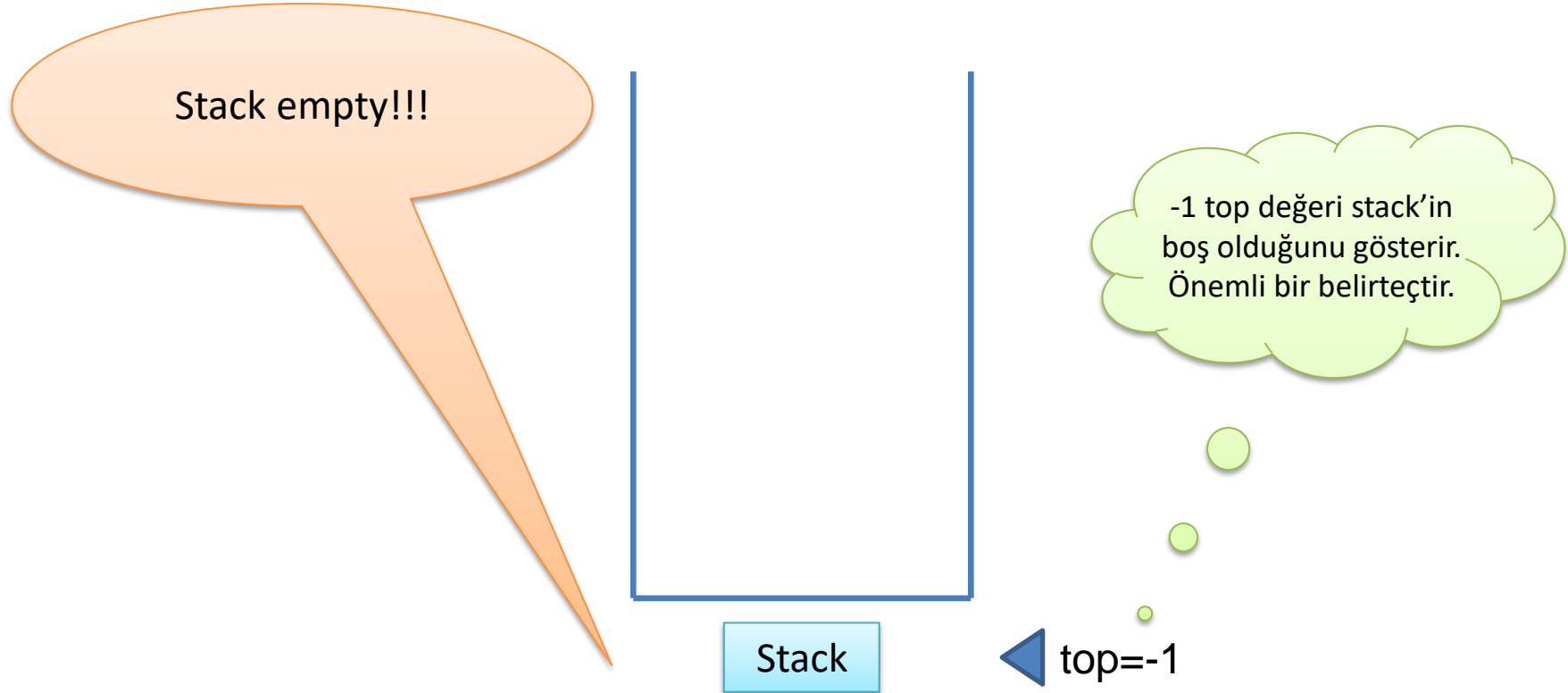
Stack Çalışma Mantığı (devam...)

- Eleman kalmadı.



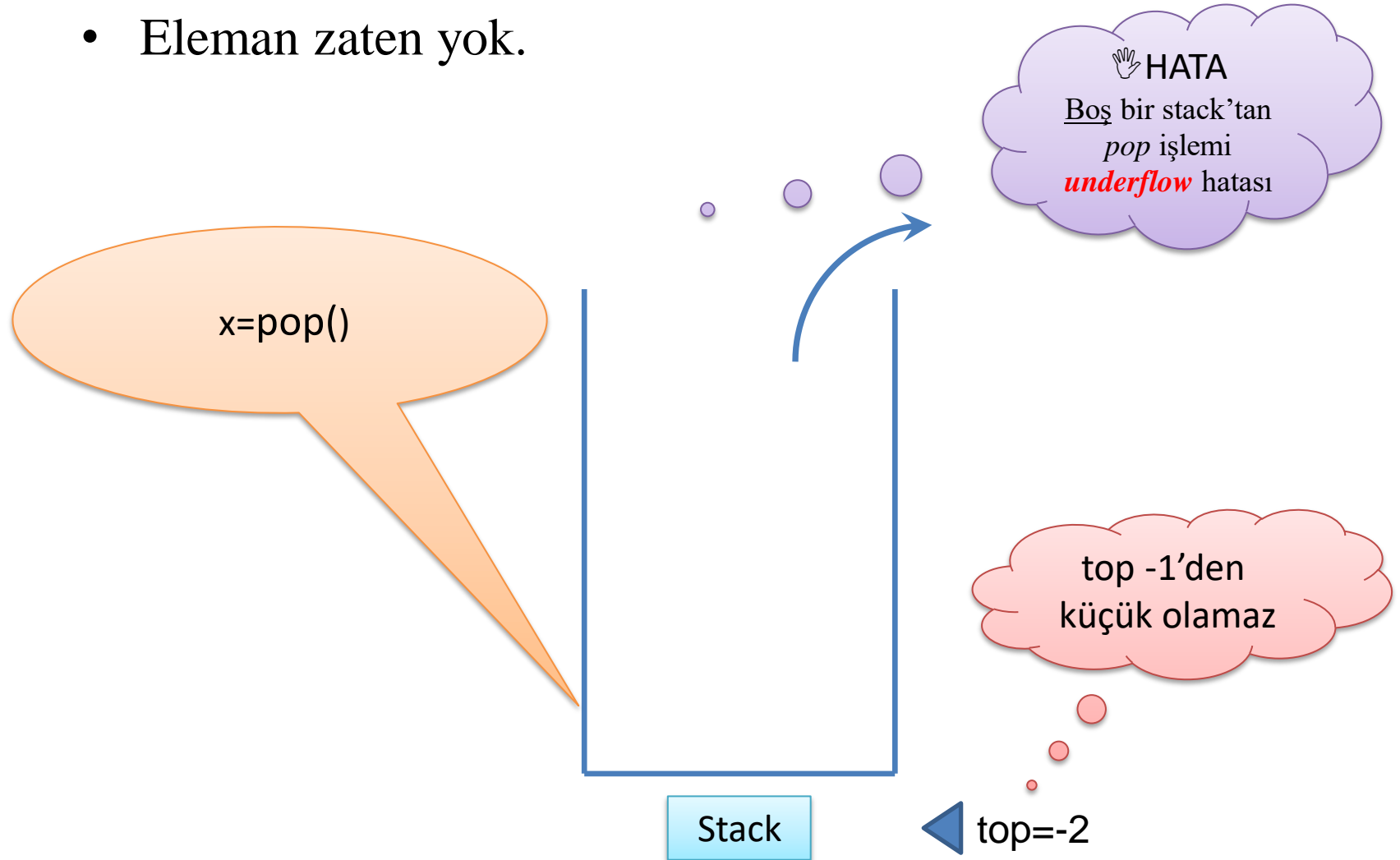
Stack Çalışma Mantığı (devam...)

- Eleman kalmadı.



Stack Çalışma Mantığı (devam...)

- Eleman zaten yok.



Stack Implementasyonu

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

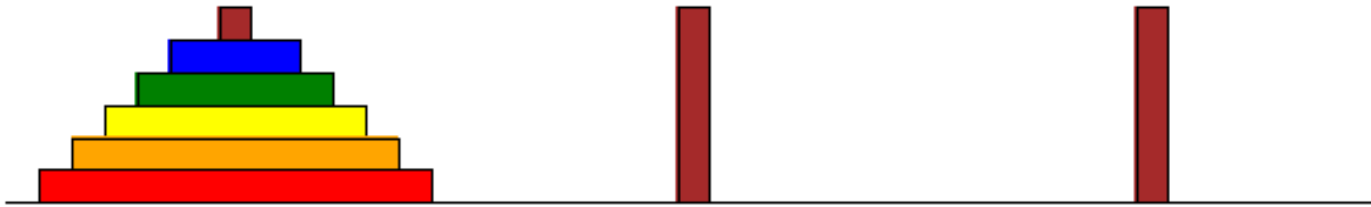
Stack Uygulamaları

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

Uygulama-Hanoi Kuleleri



Uygulama-Hanoi Kuleleri



Uygulama-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 okunur ve çıktı olarak verilir.

Polish Notasyonu

- 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 Notasyonu

Örnek: $A+B$

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

Polish Notasyonu

- **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 dönüştürdüktan (belirsizliği ortadan kaldırdıktan sonra) sonucu hesaplar : “3 2 * 5 6 * +”
- Hem **Infix \rightarrow Postfix dönüşümünde** hem de **Postfix hesaplamasında** **stack** kullanılır.

InfixToPostfix Algoritma

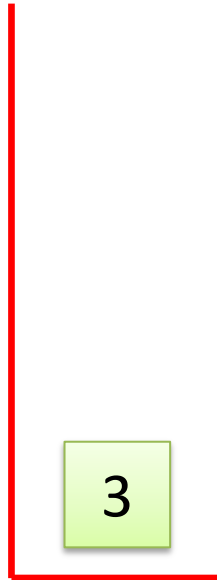
- **Sol parantez ise:** Sol parantez yığına **Push** edilir.
- **Sağ parantez ise:** Sol parantez çıkana kadar yığından **Pop** işlemi yapılır. Alınan işlem işareti **Postfix** ifadeye eklenir. Sol parantez görüldüğünde **Pop** işlemine son verilir. Sol parantez **Postfix**'e eklenmez.
- **Sayı ise:** **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.

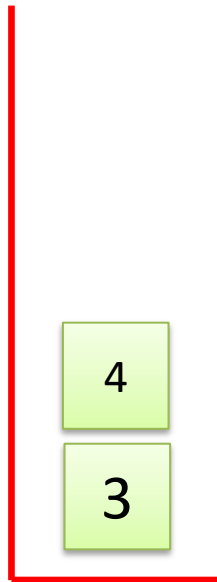
Uyg5-Postfix Değerlendirme

- Örnek: 3 4 + 5 6 * 9 2 - + *



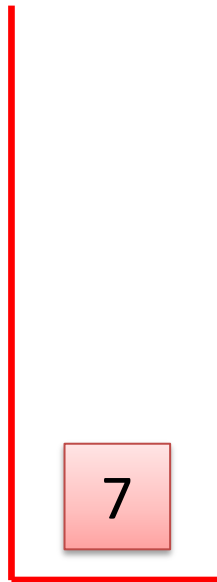
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



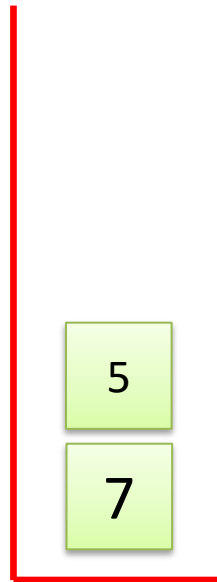
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



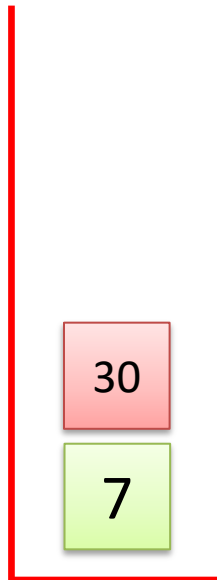
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



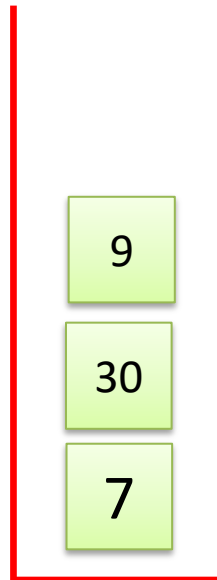
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



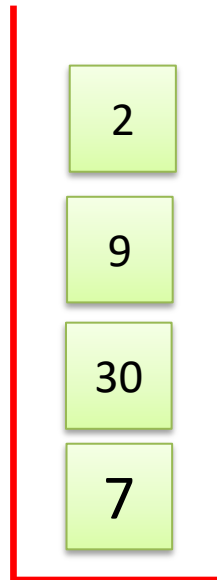
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



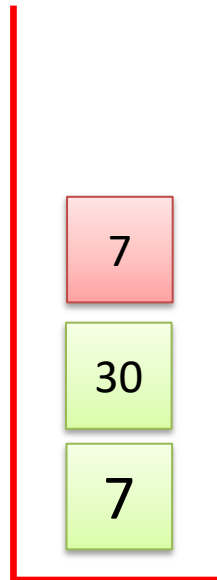
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



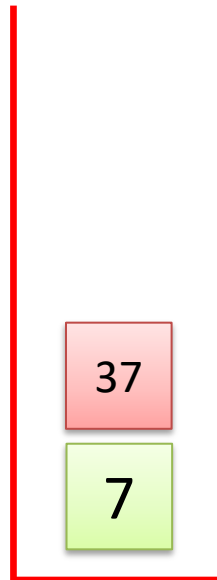
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: 3 4 + 5 6 * 9 2 - + *



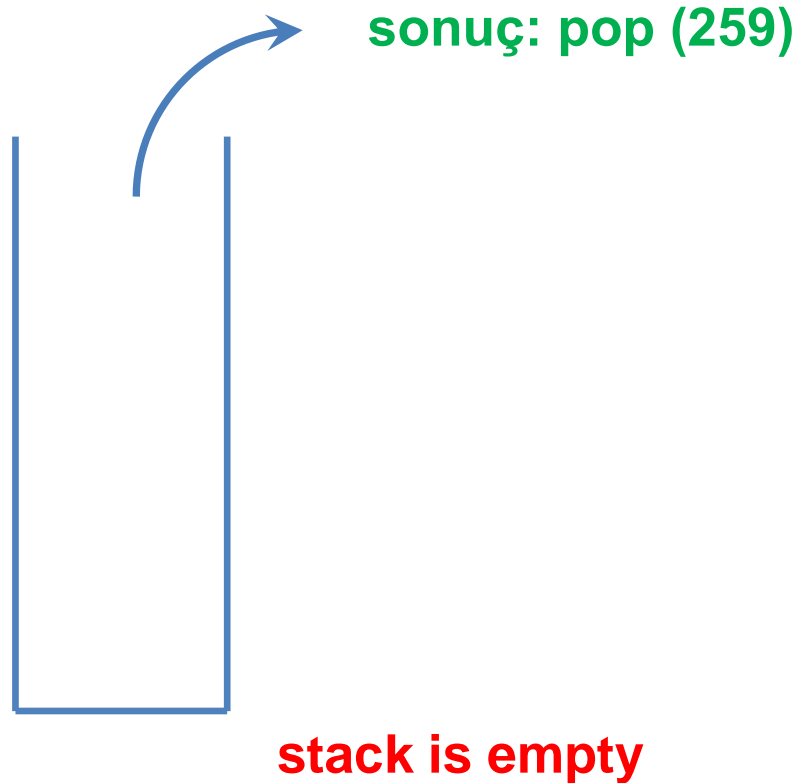
Uyg5-Postfix Değerlendirme (devam...)

- Örnek: $3\ 4\ +\ 5\ 6\ *\ 9\ 2\ -\ +\ *$



Uyg5-Postfix Değerlendirme (devam...)

- Örnek: $3\ 4\ +\ 5\ 6\ *\ 9\ 2\ -\ +\ *$



BİTTİ 😊

Yararlanılan Kaynaklar

- **Ders Kitabı:**
 - Veri Yapıları Rifat ÇÖLKESEN
 - Data Structures Using C, Reema Thareja
- **Yardımcı Okumalar:**
 - Celal Bayar Üniversitesi, Yrd. Doç. Dr. Deniz KILINÇ hocanın sunumları.