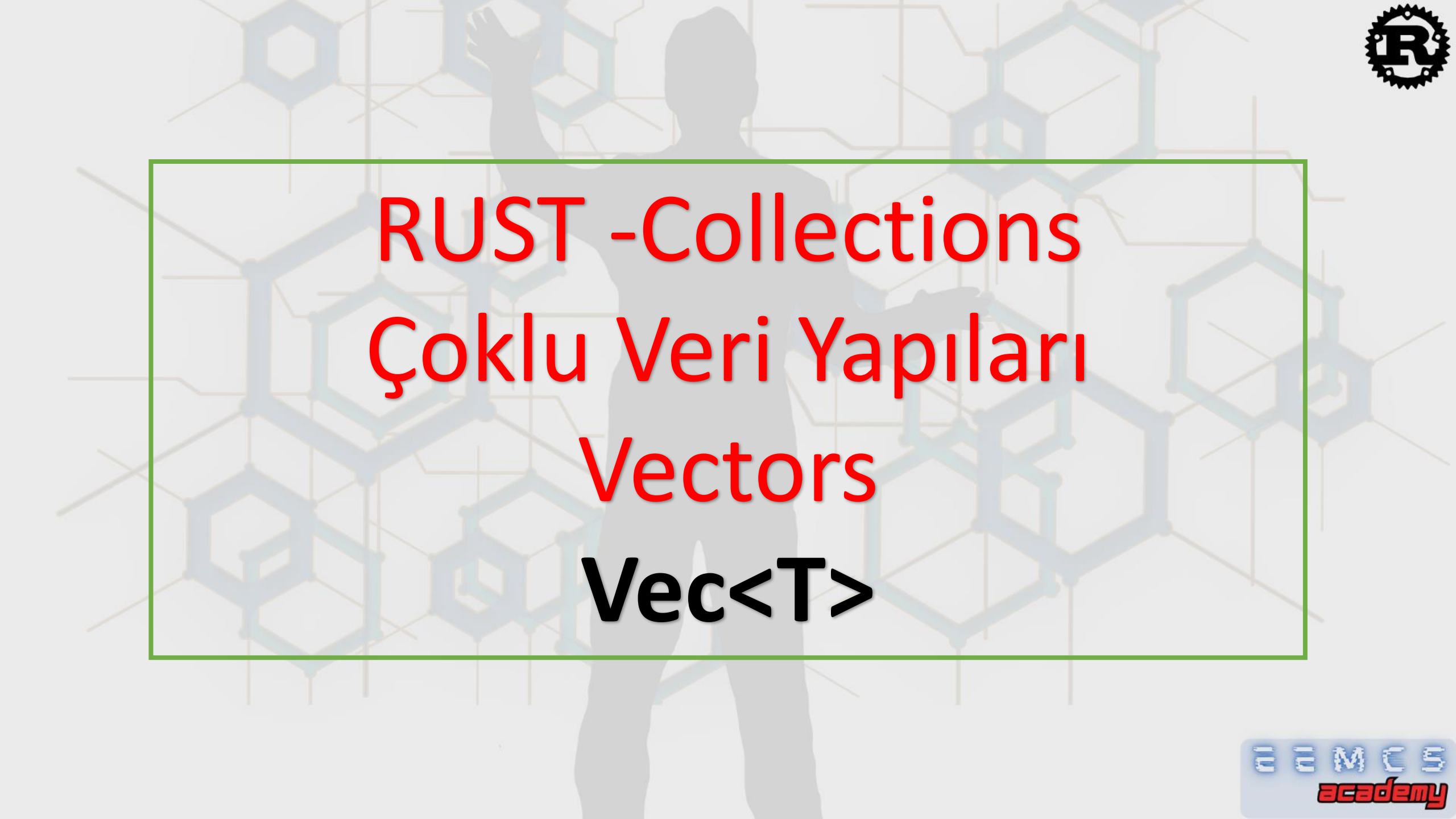

RUST -Collections
Çoklu Veri Yapıları
Vectors
Vec<T>

Çoklu Veri Yapıları

- İçerisinde birden fazla veri tutarlar
- Dizi ve demetlerden farklı olarak heap memory de saklanırlar.
- Derleme zamanında (compile time) veri miktarını bilmek gerekmez
- Veri eklenebilir-büyütülebilir ve veri çıkartılabilir - küçültülebilir

Vector → Birbirini takip eden birden fazla değer saklar
String → Daha önce gördük
Hash Map → key (anahtar) - value (değer) yapısında veri saklar. Map olarakta isimlendirilir.

Diğer veri türleri için → https://doc.rust-lang.org/std/collections/index.html

Vectors - Vec<T>

- Aynı türdeki verileri saklar
- Eleman listesi (metindeki satırlar) , alışveriş listesindeki elemanların fiyatı gibi
- Her elemanın index [ ] numarası vardır.

Vector oluşturma

Boş vektör → let vektör_adı : Vec<veri_türü> = Vec :: new();

```
let v: Vec<i32> = Vec::new();
```

// "Vec" başharfi büyük

Veri atayarak vektör oluşturma →

```
let vektör_adı = vec![veri1,veri2,veri3,….,veriN];

let v = vec![1, 2, 3];
```

// "vec" başharfi küçük

i32 türünde vectör oluşturulur. vectör_adı:Vec<u16> yazılarak veritipi kodlanabilir

Vectörün tüm elemanlarının çıktısını almak → {:?}

println!(" {:?} " , vektör_adı)

Vectör güncelleme → vectör_adı.push(eklenecek_eleman);
v.push(4);

Son elemanı silme → vectör_adı.pop();

İndex noya göre eleman silme → vektör_adı.remove(index_no);

// mut tanımlamayı unutmayın
// veri türününün aynı olmasına dikkat edin.

Vektörün Elemanlarına Erişim

İki farklı erişim türü vardır.

→ İndex numarası kullanarak

→ Option veri türü ve get() metodu kullanarak

İndex numarası kullanarak :

```
let v = vec![1, 2, 3, 4, 5];
let ucuncu: &i32 = &v[2];
println!("The third element is {third}");
```

Option Veri Türü ve get() Metodu Kullanarak:

```
let v = vec![1, 2, 3, 4, 5];

let ucuncu: Option<&i32> = v.get(2);
```

Option<T> Nedir?

Özel tanımlı, enum yapısında bir veri türüdür.
İki adet değeri vardır.

Some(T)
None

Yukarıdaki gibi, veri döndüren yapılardan güvenli veri almak için kullanılır. Yani veri geliyorsa Some(T) değerini verir; eğer veri gelmiyorsa None değerini verir.

Böylece dönen veri yoksa bile program çökmez.

Ve iki durum match ile kontrol edilerek programda güvenli akış sağlanmış olur.

İki Durum Arasındaki Farklar

```
let v = vec![1, 2, 3, 4, 5];
let olmayan_eleman = &v[100]; // HATA OLUŞUR
let olmayan_eleman = v.get(100);
```

Örnekte vektörün olmayan 99. elemanına erişilmeye çalışılmaktadır.

Birinci seçenekte program çöker, rust programlama literatüründe geçen panic durumu oluşur. Programın çökmesini istediğiniz durumlarda index numarası kullanabilirsiniz.

İki Durum Arasındaki Farklar

```
let olmayan_eleman = v.get(100);
```

İkinci seçenekte ise yani get() metodu ;

Ya bir değer döndürür Some<T>, T dönen değerin türüni temsil eder.

Ya da boş değer anlamına gelen None değerini döndürür.

Bu şeçeneği vektörde aranan değerin bazen var olabilmesi ihtimaline göre kullanabilirsiniz.

Some<T> → T : gelen verinin tipini temsil eder. Bu u16, String vb. herhangi bir veri tipi olabilir.

Some(5), Some("Rust") vb. şeklindedir.

```
let pogramlama_dili = Some("Rust");
```

Şeklinde de bir değişken tanımalaması yapılabilir. Bu verinin tipi Option<T> olur.

Gelen Veriye Göre İşlem Yapmak:

```
let v = vec![1, 2, 3, 4, 5];

let olmayan_eleman: Option<&i32> = v.get(100);

match olmayan_eleman {
      Some(eleman) => println!("Vektörden alınan öğe
{eleman}"),
      None => println!("Böyle bir vektör öğesi yoktur."),
}
```

Some(T)' Tipinden Veriyi Almak - 1

```
let v = vec!["rust", "python", "java", "go lang"];

let eleman = v.get(100);

let mut veri = String::new();

match eleman {
     Some(alınan_veri) => { println!("Vektörden alınan öğe
{alınan_veri}");
                            veri = alınan_veri.to_string();
                          },
     None => println!("Böyle bir eleman yoktur."),
}
```

Some(T)' Tipinden Veriyi Almak - 2 (sayısal tipler için)

```
let v = vec![1, 2, 3, 4, 5];

let ucuncu = vector.get(index_no).copied().unwrap();

let ucuncu =
vector.get(index_no).copied().unwrap_or(kendi_belirlediğiniz_bi
r_değer);

let ucuncu = vector.get(index_no).copied().unwrap_or_default();
//Gelecek veri tipine göre varsayılan değer atar.
```

Some(T)' Tipinden Veriyi Almak - 2 (String ve &str tipler için)

```
&str türündeki vector elemanları için

let veri= vector1.get(10).clone().unwrap_or(&"..")

String türündeki vector elemanları için

let veri=
vector1.get(10).clone().unwrap_or(&String::from("…
").to_string()
```

Vectör'de Ownership Kuralı

Mutable bir vektörde;

Bir eleman ödünç alındıktan sonra

Önce ödünç alınan verinin kullanılması gerekir

```
// HATA OLUŞUR
let mut v = vec![1, 2, 3, 4, 5];
let first = &v[0];
v.push(6);
println!("The first element is: {first}");
```

Ödünç alınan veriyi ekeleme yaptıktan sonra kullanmaya çalışırsak hata oluşur.

Çünkü owner değişmiştir.

```
// ÇALIŞIR
let mut v = vec![1, 2, 3, 4, 5];
let first = &v[0];
println!("The first element is: {first}");
v.push(6);
```

Vector Elemanları Arasında Gezinme

```
let v = vec![100, 32, 57]; // immutable

for i in &v {
     println!("{i}");
}
```

```
let mut v = vec![100, 32, 57]; // mutable

for i in &mut v {
     *i += 50;
}
```

for döngüsünün tuttuğu vektör referansı, tüm vektörün aynı anda değiştirilmesini önler. Vektöre ekleme ve çıkarma yapmaya çalışırsak çalışma zamanı hatası oluştur. For döngüsü ile vektör elemanları arasında gezinmek bu yüzden güvenlidir.

Vector'de Çoklu Veri Tiplerininin Kullanılması

Vektörler sadece aynı türdeki verileri saklar
Bu bazen dezavantajdır.
Enum veri türleri kullanarak çoklu veri tiplerinde vektör oluşturabiliriz.

Vector'de Çoklu Veri Tiplerininin Kullanılması

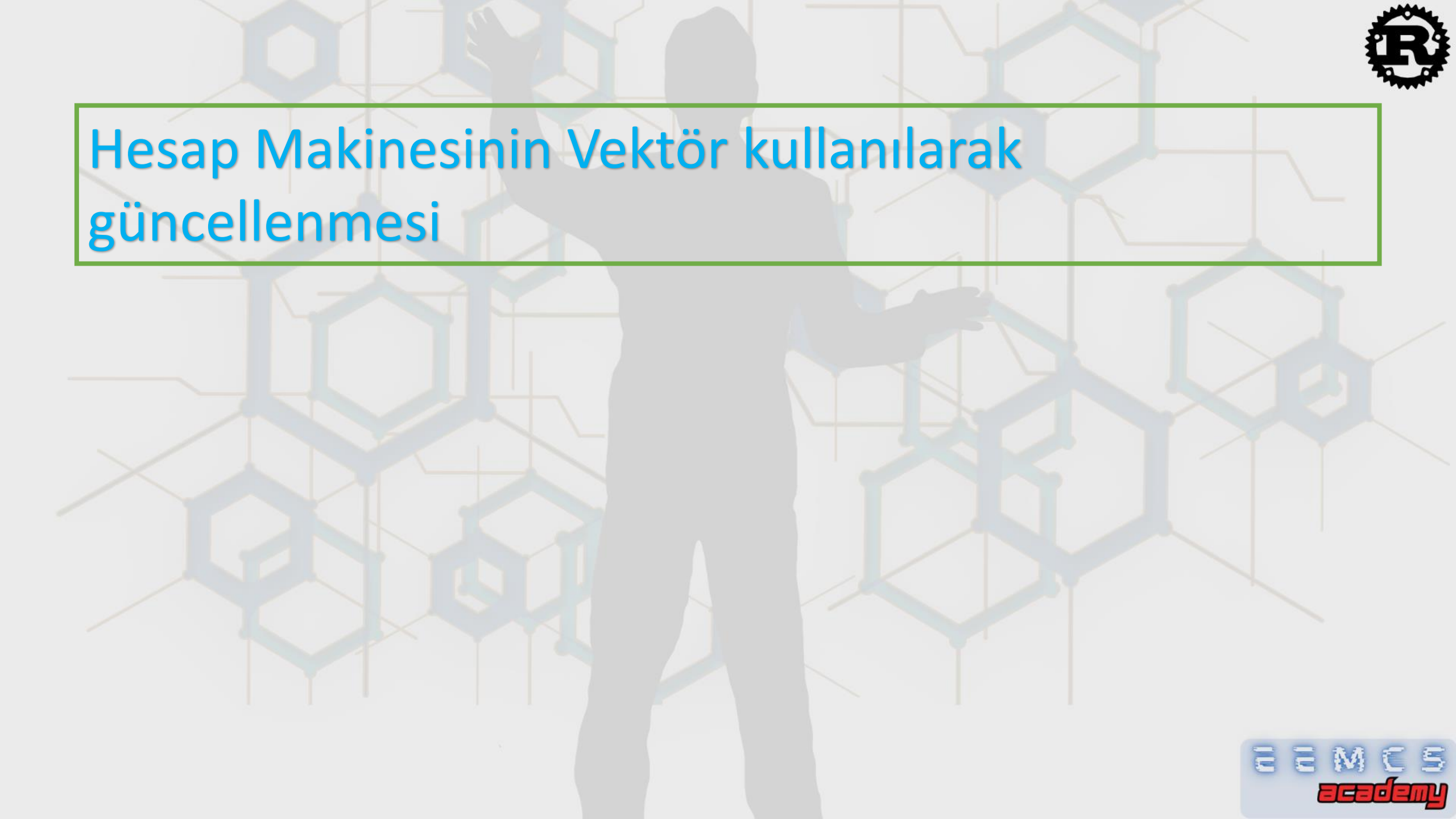
Aşağıdaki gibi bir tablonun satırlarını vector ile saklayalım

Sıra no - integer	Boy -float	İsim - text
1	1.60	Fatma

Vector'de Çoklu Veri Tiplerininin Kullanılması

```
enum SpreadsheetCell {
      Int(i32),
      Float(f64),
      Text(String), }

let row = vec![
      SpreadsheetCell::Int(3),
      SpreadsheetCell::Text(String::from("blue")),
      SpreadsheetCell::Float(10.12),
];
```



Hesap Makinesinin Vektör kullanılarak güncellenmesi

```rust
#[allow(unused_imports)]
use std::io::*;
#[allow(unused_variables)]
fn main() {
    loop {
        programa_giris();
        let secim = secim_yap();
        println!("Seçiminiz {secim}");
        let verilerim = veri_girisi();
        println!("Verileriniz {:?}", verilerim);
        match secim {
            1 => toplama(&verilerim),
            2 => cikarma(&verilerim),
            3 => carpma(&verilerim),
            4 => bolme(&verilerim),
            _ => println!("Yanlış seçim yaptınız"),
        }
        let cikis = cikis();
        if cikis.trim() == String::from("h") {
            break;
        }
    }
}
```

```rust
fn programa_giris() {
    // Kullanıcıya program ile ilgili bilgilerin gösterilmesi
    println!("Bu program basit bir hesap makinesidir.");
    let mesaj = r####"
                    Burada yapılan işlemler
                    1 - Toplama
                    2 - Çıkarma
                    3 - Çarpma
                    4 - Bölme
                "####;

    println!("{mesaj}");
}
// Kullanıcının işlem seçimi
fn secim_yap() -> u8 {
    // Kullanıcıdan yapması gereken işlemi seçme kısmı
    let mut islem = String::new();
    println!("Yapmak istediğiniz işlemin mumarasını giriniz . ( 1/2/3/4 )");
    stdin().read_line(&mut islem).expect("hata");
    let islem: u8 = islem.trim().parse().expect("hata");
    islem
}
```

```rust
fn veri_girisi() -> Vec<u32> {
    let mut veriler = Vec::new();
    let mut sayac = 1;
    loop {
        let mut kullanıcı verisi1 = String::new();
        println!(
            "{}. sayıyı giriniz (Çıkmak için q harfine basınız.) : ",
            sayac
        );
        sayac += 1;
        stdin().read_line(&mut kullanıcı verisi1).expect("hata");
        if kullanıcı verisi1.trim() == String::from("q") {
            break;
        }
        let sayı1: u32 = kullanıcı verisi1.trim().parse().expect("hata");
        veriler.push(sayı1);
    }
    println!("_fonksiyondaki Çıktı_ Girdiğiniz değerler {:?}", veriler);
    veriler
}
```

```rust
fn toplama(veriler: &Vec<u32>) {
    let islem = String::from("Toplama");
    let mut islem_sonucu = 0;
    for i in veriler {
        islem_sonucu += i; // islem_sonucu = islem sonucu + i
    }
    sonuc_cıktısı(islem_sonucu, islem);
}
// Çıkarma
fn cikarma(veriler: &Vec<u32>) {
    let islem = String::from("Çıkarma");
    let mut islem_sonucu = veriler[0];
    let mut i = 1;
    loop {
        islem_sonucu -= veriler[i];
        i += 1;
        if i == veriler.len() {
            break;
        }
    }
    sonuc_cıktısı(islem_sonucu, islem);
}
```

```rust
fn carpma(veriler: &Vec<u32>) {
    let islem = String::from("Çarpma");
    let mut islem_sonucu = veriler[0];
    for i in veriler {
        islem_sonucu *= i; // islem_sonucu = islem sonucu + i
    }
    sonuc_cıktısı(islem_sonucu, islem);
}
// Bölme
fn bolme(veriler: &Vec<u32>) {
    let islem = String::from("Bölme");
    let mut islem_sonucu = veriler[0];
    let mut i = 1;
    loop {
        islem_sonucu /= veriler[i];

        i += 1;

        if i == veriler.len() {
            break;
        }
    }
    sonuc_cıktısı(islem_sonucu, islem);
}
```

```rust
fn cikis() -> String {
    let mut cevap = String::new();
    println!("Devam etmek istiyor musunuz? Programdan çıkmak için h yazınız!");
    println!("Devam etmek için enter tuşuna basın");
    stdin().read_line(&mut cevap).expect("hata");
    cevap
}
// sonuç çıktısı
fn sonuc_cıktısı(sonuc: u32, islem: String) {
    println!("{} işleminin sonucu {} ", islem, sonuc);
}
```

Standard library de bulunan vector metodları için aşağıdaki bağlantıya bakabilirsiniz.

https://doc.rust-lang.org/std/vec/struct.Vec.html

Celal AKSU
Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

https://www.linkedin.com/in/cllaksu/

https://twitter.com/ksacll

https://www.youtube.com/@eemcs