RUST
Koşullu İfadeler
Karşılaştırma Operatörleri
Mantıksal Operatörler

Conditional Statements - Koşullu ifadeler
if / if-else / if-else if-else

Hava Durumu -

sıcak - kısa kol giyin
soğuk - kazak giyin
kış - mont giyin
sonbahar - ceket giyin

Ders durumu

başarılı - aynı plan ile çalışmaya devam et
başarısız - çalışma saatini arttır

Karşılaştırma Operatörleri

Verileri (değişkenleri) biriyle karşılaştırmak için kullanılır.
İşlemin sonucu mantıksal (bool) veri üretir (true - false)
Kontrol ifadelerinde (if) ve döngülerde (while) kullanılır

```
let sayı1 = 15;
let sayı2 = 23;
```

		Karşılaştırma	Üretilen Veri
<	→ Küçük	sayı1 < sayı2	true
>	→ Büyük	sayı1 > sayı2	false
<=	→ Küçük veya eşit	sayı1 <= sayı2	true
>=	→ Büyük veya eşit	sayı1 >= sayı2	false
==	→ Eşit	sayı1 == sayı2	false
!=	→ Eşit değil	sayı1 != sayı2	true

```
let sayı1 = 5;
let sayı2 = 10;

let kr1 = sayı1 < sayı2;
println!("{}", kr1);
```

kr1 değişkeninin **veri tipinin boolean** (true veya false) olacağını unutmayınız.

Bu karşılaştırmaları ve boolean değişkenleri **if** gibi kontrol ifadelerinde ve **while** gibi döngülerde kullanacağız

Conditional Statements - Koşullu ifadeler
if / if-else / if-else if-else

```
if bool_değişken {
        // buradaki kodlar bool_değişken true olunca çalışır
}
```

Yeni bir scope (kapsam) başladığına dikkat edilmelidir.

Yeni kapsamdaki kodlar **bool_değişken** değeri **true** olduğunda **çalışır, false** olduğunda **çalışmaz.**

```
    if kr1 {
        println!("if çalıştı");
    }
```

" bool_değişken " kısmında,
bool veri üreten herhangi bir işlemi yazabiliriz

örneğin verileri birbiri ile karşılaştırabiliriz.

```
    if sayı1 < sayı2 {
        println!("if çalıştı");
    }
```

Conditional Statements - Koşullu ifadeler
if / if-else / if-else if-else

```
if bool_değişken {
        // buradaki kodlar bool_değişken true olunca çalışır
} else {
        // buradaki kodlar bool_değişken false olunca çalışır
}
```

" else " ten sonra oluşan kapsamdaki kodlar bool_değişken verisi false olduğunda çalışır.

Kapsam içinde tanımlanan değişkenlerin kapsam dışında geçerli olmayacaklarına dikkat etmelisiniz. Bu yüzden, kapsamda üretilecek veri kapsam dışında kullanılacak ise, değişken if satırından önce tanımlanmalıdır.

```
    if kr1 {
        println!("if çalıştı");
    } else {
        println!("else çalıştı")
    }
```

```
    if sayı1 < sayı2 {
        println!("if çalıştı");
    } else {
        println!("else çalıştı")
    }
```

Bir değişkenin birden fazla değerinin kontrol edilmesi

```
if veri_kontrolü_bir {
        // veri_kontrolü_bir  true olunca çalışır. Diğerleri çalışmaz
} else if veri_kontrolü_iki {
        // veri_kontrolü_iki true olunca çalışır. İlk kontrol false olmuştur.
} else if veri_kontrolü_üç {
        // veri_kontrolü_üç true olunca çalışır. İlk iki kontrol false olmuştur
} else {
        // bütün kontroller false olunca çalışır
}
```

Bir değişkenin birden fazla değerinin kontrol edilmesi

```
if not == 1{
        // kodlar - - önceki koşullar false, bu koşul true olunca
} else if not == 2 {
        // kodlar - önceki koşullar false, bu koşul true olunca
} else if not == 3 {
        // kodlar - önceki koşullar false, bu koşul true olunca
} else if not == 4 {
        // kodlar - önceki koşullar false, bu koşul true olunca
} else {
        // KODLAR - bütün koşullar false olunca
}  // KOŞULLARDAN BİRİ ÇALIŞINCA DİĞER KOŞULLAR DEVRE DIŞI KALIR
```

```
let not = 3

1- başarısız

2-geçer

3-orta

4-iyi
```

Değişken tanımlanmasında (let) if Kullanımı

```
    let buyuk_sayi = if sayı1 > sayı2 { sayı1 } else { sayı2 };

    println!("Büyük olan sayı {}", buyuk_sayi);
```

Birden fazla koşulun birlikte kontrol edilmesi

Aynı değişkenin alabileceği farklı değerler ya da birden fazla değişkenin durumları birlikte de kontrol edilebilir.

Mantıksal Operatörler && (and) - || (or) - ! (not)

Birden fazla karşılaştırma işlemini aynı anda yapmak için kullanılır
İşlemin sonucu mantıksal (bool) veri üretir (true - false)

```
let sonuç = 5 > 6 && 7 < 10    sonuç
            f        t          f
let sonuç = 8 > 6 && 7 < 10
            t        t          t
let sonuç = 5 > 6 && 7 < 3
            f        f          f
```

&& → AND - VE : Sağında ve solundaki ifadelerin sonucu true ise üretilen veri true olur; herhangi bir false ise üretilen veri false olur.

|| → OR - VEYA : Sağında ve solundaki ifadelerden biri true ise üretilen veri true olur; bütün ifadeler false üretilen veri false olur.

```
let sonuç = 5 > 6 || 7 < 10        sonuç
            f        t             t

let sonuç = 5 > 6 || 7 < 10
            t        t             t

let sonuç = 5 > 6 || 7 < 4
            f        f             f
```

! → NOT - DEĞİL : Verilen bool verinin tersini alır; true ise false, false ise true veri üretir.

```
let sonuç = ! ( 5 > 6 )          sonuç
                f                t


let sonuç = ! ( 7 < 10 )
                t                f
```

Karşılaştırmaları ihtiyaca göre parantez () içine alarakta yazılabilir.

```
let sonuç = ( ! 5<6 ) && ( a > b || 8 != 10)      sonuç
              !(f)    &&       ( f || t)
               t      &&          t                 t
```

Birden fazla koşulun kontrol edilmesi

Bir değişkenin alabileceği değer bir den fazla olabilir

Örneğin; Puan bilgisinin alınacağı bir değişken kullandığınızı varsayın
Bu veri 0-100 aralığındadır.
Ve bu değere göre başarı durumu farklı yorumlanmaktadır.

Birden fazla koşulun kontrol edilmesi

0 - 49 → Başarısız
50 - 100 → Başarılı

Birden fazla koşulun kontrol edilmesi

```
let puan = 50;

if puan < 50 && puan >= 0 {
    println!("başarısızsın");
} else if puan > 49 && puan < 101 {
    println!(" başarılısın");
} else if puan > 100 || puan < 0 {
    println!("yanlış puan girmişsiniz")
}
```

Farklı değişkenlerin birlikte kontrol edilmesi

Örneğin bir öğrencinin başarı durumunun iki kriterden birine bağlı olduğunu düşünün.
Ya öğrenci 50 puan alacak ya da spor etkinliğinde başarılı olacak. Böyle bir durumda öğrenci puanı ve spor etkinliği birlikte kontrol edilebilir.

Farklı değişkenlerin birlikte kontrol edilmesi

```
let puan = 30;
let spor = "başarılı";

if ( puan < 50 && puan >= 0 ) && spor != "başarılı" {
    println!("başarısızsın");
} else if ( puan > 49 && puan < 101 ) || spor == "başarılı" {
    println!(" başarılısın");
}
```

İç içe koşullu ifadeler

```
if koşul1 {
        if koşul2 {
                if koşul3 {

                        .....
                }
        }
}
```

Birbirine bağımlı birden fazla değişkenin durumunu kontrol etmek kullanılır.

koşul1 true ise koşul2 kontrol edilir
koşul2 de true olursa koşul3 kontrol edilir

koşul1 false olunca diğer koşullara bakılmaz
koşul2 false olunca da koşul3 e bakılmaz

İç içe koşullu ifadeler

```
if kullanıcı_adı == "……." {
        if parola == "……." {
                if doğrulama_kodu == "……" {
                        println!("Başarılı giriş yaptınız.");
                }
        }
}
```

Hesap makinesi örneği :

```rust
use std::io::*;
#[allow(unused_variables)]
fn main() {
    let mut kullanıcı_verisi1 = String::new();
    let mut kullanıcı_verisi2 = String::new();

    println!("Bu program basit bir hesap makinesidir.");

    let mesaj = r####"
        Burada yapılan işlemler
        1 - Toplama
        2 - Çıkarma
        3 - Çarpma
        4 - Bölme
     "####;

    println!("{mesaj}");

   let mut islem = String::new();

   println!("Yapmak istediğiniz işlemin mumarasını giriniz . ( 1/2/3/4 )");

   stdin().read_line(&mut islem).expect("hata");

   let islem:u16 = islem.trim().parse().expect("hata");

    println!("Birinci sayıyı giriniz : ");

    stdin().read_line(&mut kullanıcı_verisi1).expect("hata");

    let sayı1:u32 = kullanıcı_verisi1.trim().parse().expect("hata");

    println!("İkinci sayıyı giriniz : ");

    stdin().read_line(&mut kullanıcı_verisi2).expect("hata");

    let sayı2:u32 = kullanıcı_verisi2.trim().parse().expect("hata");
```

```rust
    if islem == 1 {
        let islem_sonucu = sayı1 + sayı2;

        println!("Toplama işleminin sonucu {} ",islem_sonucu);
    } else if islem == 2 {
        let islem_sonucu = sayı1 - sayı2;

        println!("Çıkarma işleminin sonucu {} ",islem_sonucu);
    }

    if islem == 3 {
        let islem_sonucu = sayı1 * sayı2;

        println!("Çarpma işleminin sonucu {} ",islem_sonucu);
    }

    if islem == 4 {
        let islem_sonucu = sayı1 / sayı2;

        println!("Bölme işleminin sonucu {} ",islem_sonucu);
    }


}
```

Basit Bir Hesap Makinesi (ilk hali)

```
#[allow(unused_imports)]
use std::io::*;


#[allow(unused_variables)]
fn main() {
    let mut kullanıcı_verisi1 = String::new();
    let mut kullanıcı_verisi2 = String::new();

    println!("Bu program toplama işlemi yapmaktadır.");

    println!("Birinci sayıyı giriniz : ");

    stdin().read_line(&mut kullanıcı_verisi1).expect("hata");

    let sayı1:u32 = kullanıcı_verisi1.trim().parse().expect("hata");

    println!("İkinci sayıyı giriniz : ");

    stdin().read_line(&mut kullanıcı_verisi2).expect("hata");

    let sayı2:u32 = kullanıcı_verisi2.trim().parse().expect("hata");

    // let toplama_sonucu = sayı1 + sayı2;

    println!("Toplama işleminin sonucu {} ",sayı1 + sayı2);

}
```

match ile akış kontrolü

Bir değişkenin alabileceği değer birden fazla ise ve bu değerlere göre yapılacak olan işlemlerin farklı olması durumlarında match ile akış kontrolü yapılabilir.

```
match değişken {

        değer1 => {
                // kodlar
        },
        değer2 => {
                // kodlar
        },
        .....
        _ => {

        }
}
```

Kod kısmı tek satırdan oluşuyorsa güzel parantezleri kullanmaya gerek yoktur.

```rust
let not:u8 = 3;
match not {
    1 => {
        println!("Durumunuz : Başarısız");
        println!("Daha çok gayret etmemlisiniz.")
    },
    2 => println!("Durumunuz : Orta"),
    3 => println!("Durumunuz : İyi"),
    4 => println!("Durumunuz : Pekiyi"),
    _ => print!("Notunuzu kontrol ediniz {}", not)
}
```

```
match değişken {
        .
        .
        .
        değer_n => {
                // kodlar
        },
        diger_degerler => {
                // kodlar
                println!("Değişkenin değeri {}",diger);
        }
}
```

Kontrol edilen değerlerin dışındaki değerler farklı bir değişkende de tutulabilir. Burada kullanılan değişken adı diğer_degerler olarak verilmiştir.

| (veya) ile birde fazla değerin birlikte kontrol edilmesi

```
let x = 1;

match x {
    1 | 2 => println!("one or two"),
    3 => println!("three"),
    _ => println!("anything"),
}
```

..= ile değer aralığı verilmesi

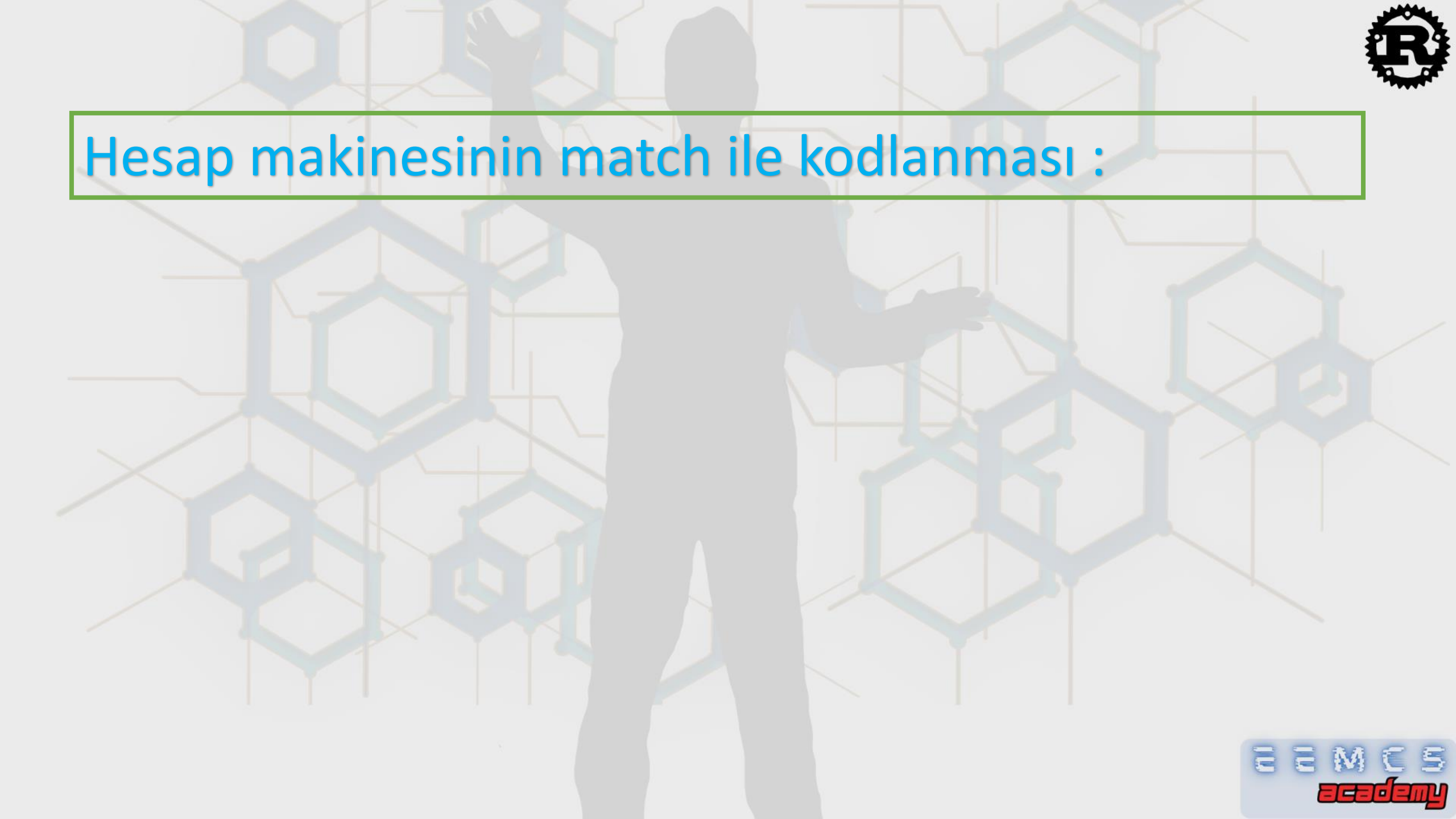
```
let x = 5;

match x {
    1..=5 => println!("one through five"),
    _ => println!("something else"),
}
```

..= ile değer aralığı verilmesi

```
let x = 'c';

match x {
    'a'..='j' => println!("early ASCII letter"),
    'k'..='z' => println!("late ASCII letter"),
    _ => println!("something else"),
}
```



Hesap makinesinin match ile kodlanması :

Celal AKSU
Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

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

https://twitter.com/ksacll

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