
RUST
Fonksiyonlar - Functions
fn

Fonksiyon nedir?

Programı küçük parçalara bölmeyi sağlar
Hata bulmayı kolaylaştırır
Pogramı geliştirmeyi kolaylaştırır
Programa yeni eklemeler yapmayı kolaylaştırır.
Paket, modül, kütüphane vb. yapıların en basit hali gibi düşünebiliriz.

main() Fonksiyonu

Pek çok programın giriş noktasıdır; (Diğer bir değişle programın yönetim konsoludur)

program çalıştırıldığında sadece buradaki kodlar çalışır
bu fonksiyon dışındaki kodlar çalışmaz, dead_code olarak geçer
bu fonksiyon dışındaki kodların çalıştırılması için main() fonksiyonunda işlenmeleri gerekir

fn → Fonksiyon oluşturmak

fn fonksiyon oluşturmak için kullanılır.
main() dışında oluşturulurlar. Önce veya sonra olabilir

```
fn fonksiyon_adı() {
        // kodlar
}
```

Fonksiyon adı için küçük harfler kullanılır ve kelimelerin arasına _ (alt tire) eklenir. Bu kullanım şekli **snake case** stili olarak isimlendirilir.

Fonksiyon oluşturmak - tanımlamak

```
fn ilk_fonksiyon(){

    println!("bu ilk fonskiyonum")

}
```

Fonksiyonun çalıştırılması - çağrılması

main() fonksiyonunda adının yazılması gerekir

```
fn main() {
        .
        .
        .
        fonksiyon_adı();
        .
        .

}
```

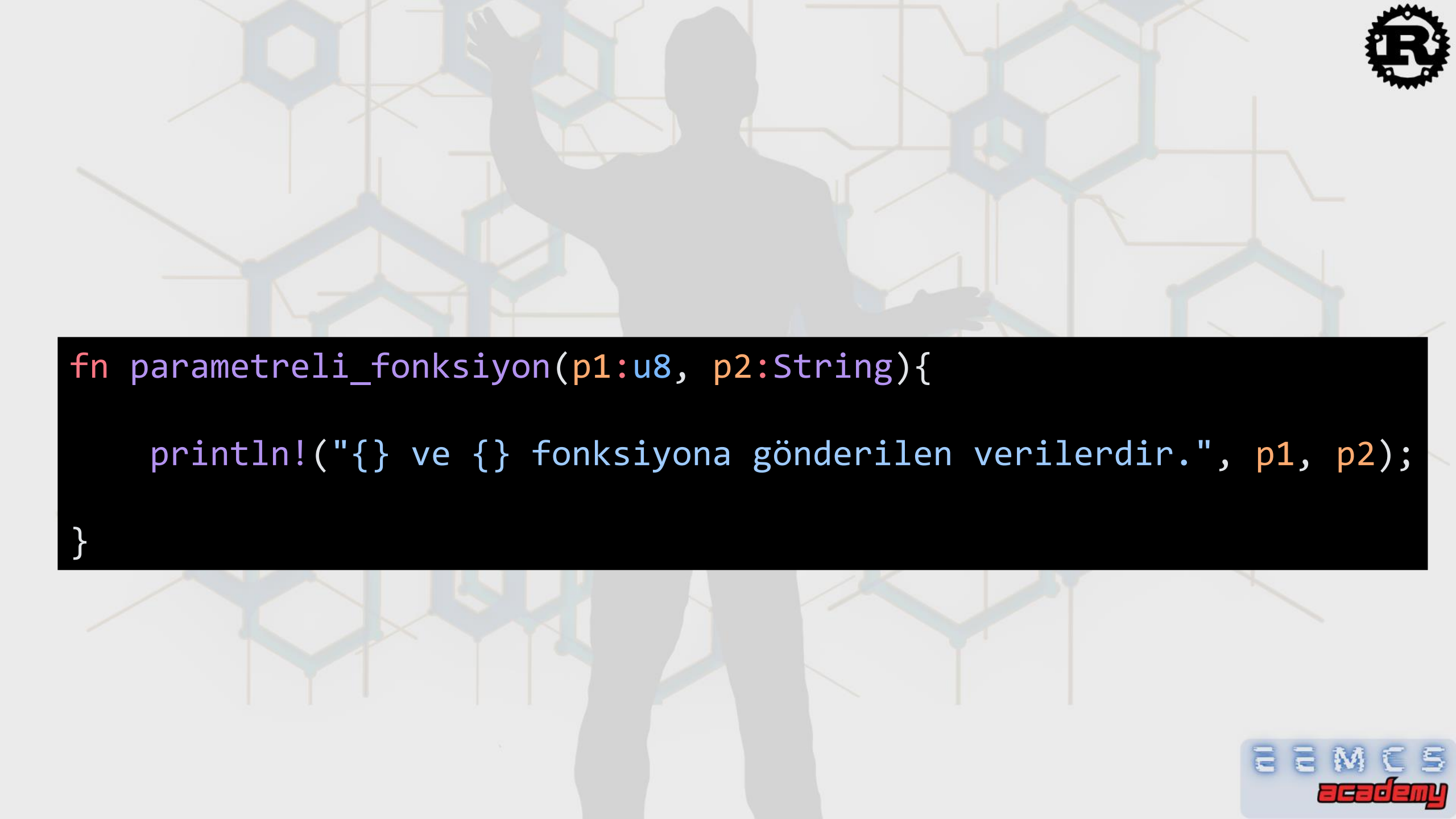
```
fn main() {

    ilk_fonksiyon();

}

fn ilk_fonksiyon(){
    println!("bu ilk fonskiyonum")
}
```

Parametreli fonksiyonlar

Parametre → Fonksiyona gönderilen verilerdir. Fonksiyon içinde bu veriler üzerinde işlemler yapabiliriz.

Birden fazla olabilir. Aralarına virgül eklenir.
() içersine yazılırlar
Farklı veya aynı değişken adını kullanabiliriz.

```
fn fonksiyon_adı(p1:veri_türü, p2:veri_türü, … ) {
        // fonksiyona gönderilen veriler parametre kısmında verilen değişken isimleri kullanarak işlenir
}
```



```
fn parametreli_fonksiyon(p1:u8, p2:String){

    println!("{} ve {} fonksiyona gönderilen verilerdir.", p1, p2);

}
```

Parametreli fonksiyonu çalıştırmak

Fonksiyondaki parametre sayısı kadar veri gönderilmelidir.
Gönderilecek değişkenler, () içine, sadece değişken isimleri yazılarak gönderilir.
Gönderilen verilerin veri türü ile parametrelerdeki veri türleri aynı olmalıdır
Gönderilen veriler sırası ile parametrelere aktarılır

```
fn main() {

    let sayi:u8 = 8;
    let metin = String::from("Rust");

    parametreli_fonksiyon(sayi, metin);

}
```

sayi değişkeninin değerinin kopyası → p1' atanır
metin değişkeninin değeri → p2'ye taşınır (move).

Değiştirilebilir - mutable - parametreli fonksiyonlar

```
fn fonksiyon_adı( mut p1:veri_türü, p2:veri_türü, … ) {
        // kodlar
}
```

Değer döndüren fonksiyonlar - return

Fonksiyonda üretilen bir verinin;
ana programa main() fonksiyonuna aktarma işlemidir
ya da verinin fonksiyonda kullanılabilir hale getirilmesi de diyebiliriz.

Fonksiyon tanımlanırken **() lerden sonra ve { önce -> veri_türü** yapısı kullanılarak fonksiyonun döndüreceği veri türü belirtilir.

```
fn deger_donduren_fonksiyon() -> u16 {
```

Fonksiyon içinde return veri/değişken_adı ifadesi kullanılarak işlem tamamlanır.

```
    return 350
    return değişken_adı
}
```

Değer döndüren fonksiyonu çağırma

Fonksiyon main()'de çağrılırken, fonksiyon tarafından döndürülen verinin alınması için fonksiyon değişkene atanarak çağrılır.

```
fn main() {

    let gelen_deger = deger_donduren_fonksiyon();
    println!("Fonksiyondan gelen değer : {}",gelen_deger);

}
```

Değer döndüren fonksiyonlar - return

return kullanımı zorunlu değildir
Verinin kendisi yazılarakta aynı işlem yapılmış olur
Fakat sonuna ; konulmamalıdır.

```
fn deger_donduren_fonksiyon() -> u16 {
    350
    // 350; → Hata verir
}
```

Örneğimizde yok fakat parametreli fonksiyonlarda da aynı yapıyı kullanarak değer döndürebilirler.

Fonksiyonlar ve Parametre Verilerinin Tekrar Kullanılması (Ownership & Borrowing)

```
fn main() {

    let sayi:u8 = 8;
    let metin = String::from("Rust");

    parametreli_fonksiyon(sayi, metin);

    println!("{}", metin); // HATA VERİR. ÇÜNKÜ metin DEĞİŞKENİ move İŞLEMİ İLE p2 PARAMETRESİNE AKTARILMIŞTIR

}
```

Hatanın giderilmesi ve metin değişkeninin fonksiyona gönderildikten sonra main() fonksiyonunda tekrar kullanılabilmesi için ödünç (borrowing) alınarak gönderilmesi gerekir.

Bunun için parametre & ile veriyi ödünç alacak şekilde düzenlenmelidir.

```
fn parametreli_fonksiyon(p1:u8, p2:&String){
    println!("{} ve {} fonksiyona gönderilen verilerdir.", p1, p2);
}
```

```
fn main() {
    . . .
    parametreli_fonksiyon(sayi, &metin);

    println!("{}", metin);

}
```

Ödünç alınan verinin fonksiyonda değiştirilmesi

```
fn parametreli_fonksiyon(p1:u8, p2:&mut String){
    println!("{} ve {} fonksiyona gönderilen verilerdir.", p1, p2);
    p2.push_str(" Öğreniyorum");
    println!("{}",p2);
}
```

```
fn main() {

    let sayi:u8 = 8;
    let mut metin = String::from("Rust");

    parametreli_fonksiyon(sayi, &mut metin);

    println!("{}", metin);

}
```

Fonksiyondan Referans Geri Döndürme - Dangling References (Sarkan referanslar)

Eğer bir veriye referansınız varsa, derleyici veriye yapılan referanstan önce verinin kapsam dışına çıkmamasını sağlar. Aksi halde hata oluşur.

```
fn main() {
    let reference_to_nothing = dangle();
}

fn dangle() -> &String {
    let s = String::from("hello");
    &s
}
```

Referans Kurallarını Tekrar Hatırlıyalım :

- Herhangi bir zamanda, tek bir değiştirilebilir referansa ya da istediğiniz sayıda değiştirilemez referansa sahip olabilirsiniz.
- Referanslar her zaman geçerli olmalıdır.

Hesap Makinesi Programının Fonksiyonlarla Yeniden Kodlanması :

```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}");
}
```

```rust
// 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() -> [u32; 2] {
    // Kullanıcı veri girişinin yapılması

    let mut veriler = [0, 0];

    let mut kullanıcı_verisi1 = String::new();
    let mut kullanıcı_verisi2 = String::new();
    println!("Birinci sayıyı giriniz : ");

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

    let sayı1: u32 = kullanıcı_verisi1.trim().parse().expect("hata");
    veriler[0] = sayı1;

    println!("İkinci sayıyı giriniz : ");

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

    let sayı2: u32 = kullanıcı_verisi2.trim().parse().expect("hata");
    veriler[1] = sayı2;

    veriler
}
```

```rust
// Toplama
fn toplama(veriler: [u32; 2]) {
    let islem = String::from("Toplama");
    let islem_sonucu = veriler[0] + veriler[1];
    //println!("Toplama işleminin sonucu {} ", islem_sonucu);
    sonuc_cıktısı(islem_sonucu, islem);
}

// Çıkarma
fn cikarma(veriler: [u32; 2]) {
    let islem = String::from("Çıkarma");
    let islem_sonucu = veriler[0] - veriler[1];
    //println!("Toplama işleminin sonucu {} ", islem_sonucu);
    sonuc_cıktısı(islem_sonucu, islem);
}
```

```rust
// Çarpma
fn carpma(veriler: [u32; 2]) {
    let islem = String::from("Çarpma");
    let islem_sonucu = veriler[0] * veriler[1];
    //println!("Toplama işleminin sonucu {} ", islem_sonucu);
    sonuc_cıktısı(islem_sonucu, islem);
}

// Bölme
fn bolme(veriler: [u32; 2]) {
    let islem = String::from("Bölme");
    let islem_sonucu = veriler[0] / veriler[1];
    //println!("Toplama işleminin sonucu {} ", islem_sonucu);
    sonuc_cıktısı(islem_sonucu, islem);
}
```

```rust
// Çıkış
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);
}
```

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

Celal AKSU
Bilişim Teknolojileri Öğretmeni

celalaksu@gmail.com

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

https://twitter.com/ksacll

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