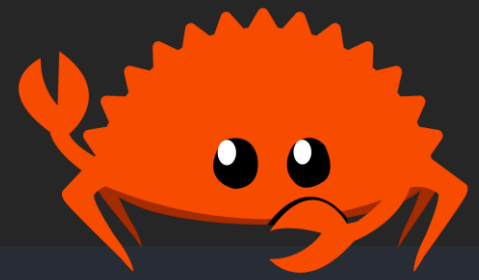
Rust Programlama Dili

Hazırlayan: Orkun Manap

Rust Programlama Dili Nedir?

moz://a

Herkesin, verimli ve güvenilir bir yazılım üretmesine izin veren bir dil.



```rust
fn main() {
    println!("Merhaba, {}", "Dünya!");
}
```

Neden Rust?

Performans

Rust, son derece hızlıdır ve bellek açısından verimlidir: çalışma zamanı (runtime) veya çöp toplayıcı (garbage collector) olmadan, performans açısından kritik hizmetlere güç sağlayabilir, gömülü cihazlarda çalışabilir ve diğer dillerle kolayca entegre olabilir.

Verim

Rust harika bir dökümantasyona, kullanışlı hata mesajlarına sahip kolay bir derleyiciye ve birinci sınıf araçlara sahiptir - entegre bir paket yöneticisi ve oluşturma (build) aracı, otomatik tamamlama ve tür denetimleriyle akıllı çoklu düzenleyici desteği, otomatik biçimlendirici ve daha fazlası.

Güvenilirlik

Rust'ın zengin tür sistemi (type system) ve sahiplik (ownership) modeli, bellek güvenliğini ve iş parçacığı (thread) güvenliğini garanti ederek derleme zamanında (compile time) birçok hata sınıfını ortadan kaldırmanıza olanak tanır.

```javascript
function logElements(arr) {
  while (arr.length > 0) {
    console.log(arr.shift());
  }
}

function main() {
  const arr = ['banana', 'orange', 'apple'];

  console.log('Before sorting:');
  logElements(arr);

  arr.sort(); // changes arr

  console.log('After sorting:');
  logElements(arr); // (A)
}
main();

// Output:
// 'Before sorting:'
// 'banana'
// 'orange'
// 'apple'
// 'After sorting:'
```

JavaScript

Bilgisayar Programlamada Bir Sorun

Shared Mutable State

Diğer Dillerden Farkı Ne?

Fonksiyonel Diller

Nesne Tabanlı Diller

Diğer Dillerden Farkı Ne?

Daha fazla kontrol,
Daha az güvenlik

Daha az kontrol,
Daha fazla güvenlik

Peki ya Rust?

Daha fazla kontrol,
Daha fazla güvenlik

Diğer Dillerden Farkı Ne?

	Avantajları	Dezavantajları
Garbage Collection	• Hatasız • Daha hızlı kodlama	• Hafıza üzerinde kontrol yok • Daha yavaş ve tahmin edilmesi güç runtime performansı
Manual Memory Management	• Hafıza üzerinde kontrol • Daha hızlı runtime • Daha düşük program boyutu	• Hata çıkma olasılığı yüksek • Daha yavaş kodlama
Ownership Model	• Hafıza üzerinde kontrol • Daha hızlı runtime • Hatasız • Daha düşük program boyutu	• Daha yavaş kodlama • Adapte olmanın zor olması (Learning curve)

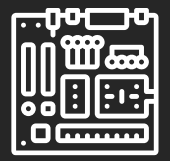
Rust ile neler yapılabilir?

Komut Satırı

Kullanıcı arayüzüne sahip olmayan komut satırı programları fonksiyonelliği ve esnekliği göz alır.

Ağ Yapıları

Üretime hazır frameworkler ile yüksek performans, az kaynak tüketimi ve sağladığı güvenlik ile ağ servisleri için harika.



Web Assembly

C++ veya Rust gibi diller ile yazdığın kodu derleyip web assembly haline getirebilir ve bunu tarayıcında JavaScript kodunun yanında çalıştırabilirsin.

Gömülü Sistemler

Gömülü cihazlar belirli amaçlar için üretilirler. Rust az kaynak tüketimiyle, vaat ettiği avantajlar ile öne çıkıyor.

Rust'ı Kimler Kullanıyor?

mozilla

Servo tarayıcı motorunda, Firefox'a entegrede, diğer projelerde.

ATLASSIAN

Petabaytlarca kaynak kodunu analiz etmek için bir hizmette.

coursera

Güvenli Docker Konteynerlerinde Programlama Atamaları

system76

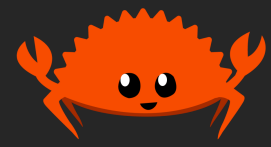
Linux tabanlı bir bilgisayar üreticisi olarak, altyapı ve masaüstü Linux projelerimizin çoğu Rust ile yazılmıştır. Donanım sertifikasyonu, yanıp sönme ve görüntülemeden; sistem hizmetlerine ve GTK3 masaüstü uygulamalarına.

CANONICAL

Sunucu izlemeden ara katman yazılımına kadar her şey!

Dropbox

Bulut dosya depolamayı optimize etmekte.



Cargo

Cargo, Rust’ın paket yöneticisidir. Cargo, Rust projenizin bağımlılıklarını indirir, projenizi derler, dağıtılabilir bir proje haline getirir ve bunları Rust topluluğunun paket kaydı olan crates.io'ya yükler.

Rust Syntax

Programlama Konseptleri

1. Değişkenler ve Değişebilirlik
2. Veri Tipleri
3. Fonksiyonlar
4. Kontrol Akışları

1. Değişkenler ve Değişebilirlik

Mutable & Immutable

```
fn main() {
    let x = 5;
    println!("The value of x is: {}", x);
    x = 6;
    println!("The value of x is: {}", x);
}
```

Shadowing

```
fn main() {
    let x = 5;

    let x = x + 1;

    let x = x * 2;

    println!("The value of x is: {}", x);
}
```

2. Veri Tipleri

Tam Sayı

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Gerçel Sayı

```
fn main() {
    let x = 2.0; // f64

    let y: f32 = 3.0; // f32
}
```

Karakter Tipi

```
fn main() {
    let c = 'z';
    let z = 'Z';
    let heart_eyed_cat = '😻';
}
```

Array

```
let a: [i32; 5] = [1, 2, 3, 4, 5];
```

```
let a = [3; 5];
```

→ let a = [3, 3, 3, 3, 3];

Tuple

```
fn main() {
    let x: (i32, f64, u8) = (500, 6.4, 1);

    let five_hundred = x.0;

    let six_point_four = x.1;

    let one = x.2;
}
```

3. Fonksiyonlar

```rust
fn main() {
    let x = plus_one(5);
    let y = five();

    println!("The value of x is: {}, y is: {}", x, y);
}

fn plus_one(x: i32) -> i32 {
    x + 1
}

fn five() -> i32 {
    5
}
```

4. Kontrol Akışı – If Else

```rust
use std::io;

fn main() {
    println!("Please input your number: ");

    let mut number = String::new();

    io::stdin().read_line(&mut number).expect("Failed to read line");

    let number: u32 = match number.trim().parse() {
        Ok(num) => num,
        Err(_) => {
            println!("Enter a number!");
            return;
        }

    };

    let result = is_divisible(number);

    // Return values must be same      You, 6 days ago • control_flow Update
    let im_just_testing = if result { "WOW!" } else { "meh." };

    println!("{}", im_just_testing);
}
```

```rust
fn is_divisible(x: u32) -> bool{

    if x % 3 == 0 && x % 2 == 0{
        print!("{}", "number is divisible by 3 and 2 ".replace('\n', " "));
        return true;
    } else if x % 3 == 0 {
        print!("{}", "number is divisible by 3 ".replace('\n', " "));
        return false;
    } else if x % 2 == 0 {
        print!("{}", "number is divisible by 2 ".replace('\n', " "));
        return false;
    } else {
        print!("{}", "nope ".replace('\n', " "));
        return false;
    }
}
```

4. Kontrol Akışı – Döngüler

```rust
// While True
fn just_loop() {
    let mut counter = 0;

    let result = loop {
        counter += 1;

        if counter == 10 {
            break counter * 2;      You, 6 days ago • init loops
        }
    };

    println!("The result is {}", result);

    //loop {
    //    println!("Hey, I am just a loop!");
    //}
}
```

```rust
fn just_while() {
    let a = [10, 20, 30, 40, 50];
    let mut index = 0;

    while index < 5 {
        println!("the value is: {}", a[index]);

        index += 1;
    }
}

fn just_for() {
    let a = [12, 23, 34, 45, 56];

    for element in a.iter() {
        println!("the value is: {}!", element);
    }

    for number in (1..4).rev() {
        println!("{}!", number);
    }
    println!("LIFTOFF!!!");
}
```

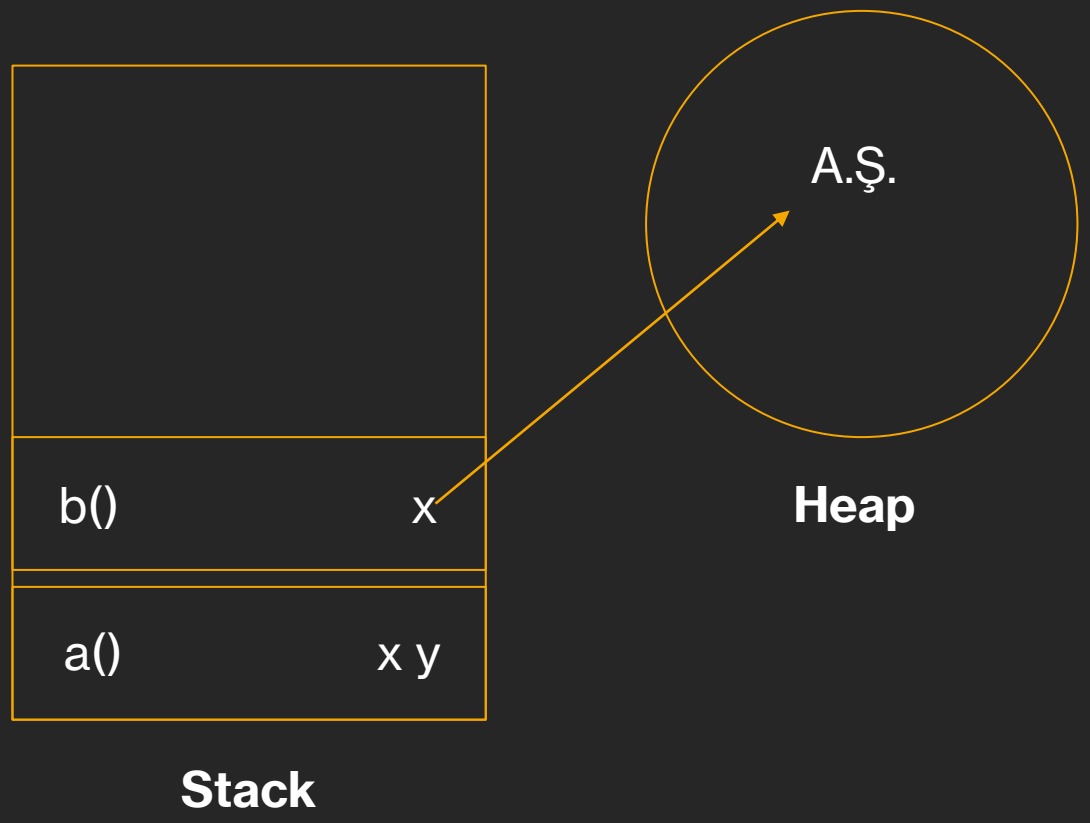
Rust Sahiplik (Ownership)

Sahipliği Anlamak

- Sahiplik nedir?
- Referanslar ve ödünç almak (borrowing)
- Dilimleme (Slice)

Stack & Heap

```rust
fn main () {
    fn a() {
        let x: &str = "HAVELSAN";
        let y: i32 = 22;

        b();
    }

    fn b () {
        let x: String = String::from("A.Ş.");
    }

}
```



Sahiplik Kuralları

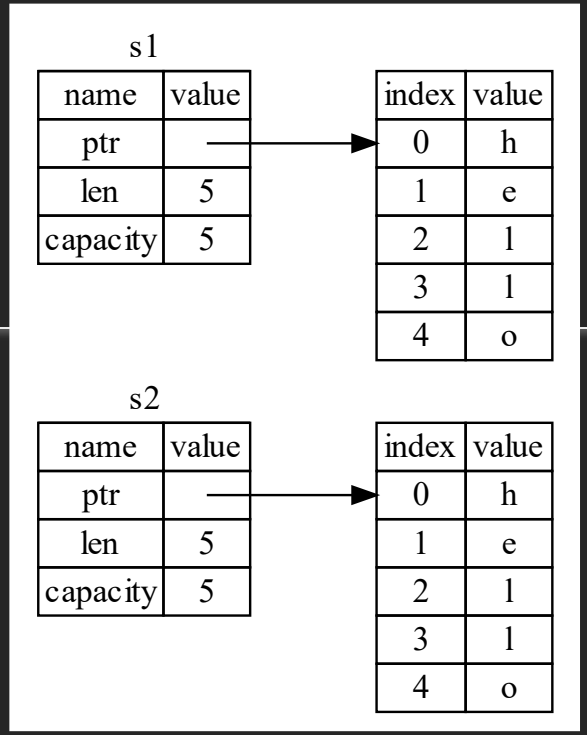
- Her bir değerin Sahip denilen bir değişkeni vardır.
- Aynı anda yalnızca tek bir sahip olabilir.
- Sahip, scope'dan çıktığında değer bırakılır (drop edilir).

```
{                       // s is not valid here, it's not yet declared
    let s = "hello";    // s is valid from this point forward

    // do stuff with s
}                       // this scope is now over, and s is no longer valid
```

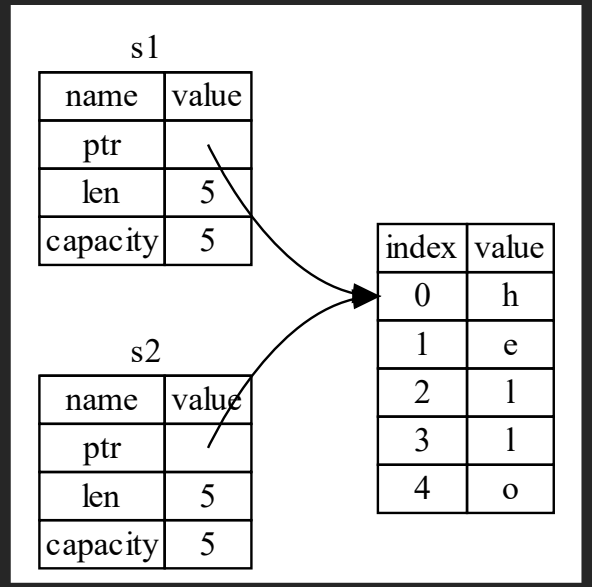
1.1 Sahiplik

```rust
fn main() {
    let x: i32 = 5;
    let y: i32 = x; // Copy

    let s1: String = String::from("Hello");
    let s2: String = s1; // Move

    println!("x: {}, y: {}", x, y);

    // It will give 'value borrowed here after move' error
    // because s1 not exists anymore
    println!("s1: {}, s2: {}", s1, s2);

    // To prevent this error, .clone() method which copies
    // can be used.
}
```

```
error[E0382]: borrow of moved value: `s1`
  --> src\main.rs:12:32
   |
5  |     let s1: String = String::from("Hello");
   |         -- move occurs because `s1` has type `String`, which does not implement the `Copy` trait
6  |     let s2: String = s1; // Move
   |                      -- value moved here
...
12 |     println!("s1: {}, s2: {}", s1, s2);
   |                                 ^^ value borrowed here after move

error: aborting due to previous error
```



Yeni Değer Oluşturarak Kopyalama



Üstünkörü Kopyalama

1.2 Sahiplik

```rust
fn main() {
    let x: i32 = 5;
    make_copy(x);
    println!("{}", x);

    let s: String = String::from("hello");
    takes_ownership(s);

    // s is borrowed
    println!("{}", s);
}

fn takes_ownership(a_string: String){
    println!("{}", a_string);
}

fn make_copy(a_int: i32){
    println!("{}", a_int);
}
```

1.3 Sahiplik

```rust
fn main() {
    let s1: String = give_ownership();
    let s2: String = String::from("Hello");
    let s3: String = take_give_ownership(s2);

    println!("s1 = {} \n s3 = {}", s1, s3);
}

fn give_ownership() -> String {
    let some_string: String = String::from("hello");

    some_string
}

fn take_give_ownership(a_string: String) -> String {
    println!("I got and gave back {}", a_string);

    a_string
}
```

1.4 Sahiplik

```rust
fn main() {
    let s1: String = String::from("hello");
    let len = calculate_length(&s1); // Sending the reference of s1 (Borrowing)
    println!("The length of '{}' is {}.", s1, len);
}

fn calculate_length(s: &String) ->  usize{
    let length: usize = s.len();
    length
}
```

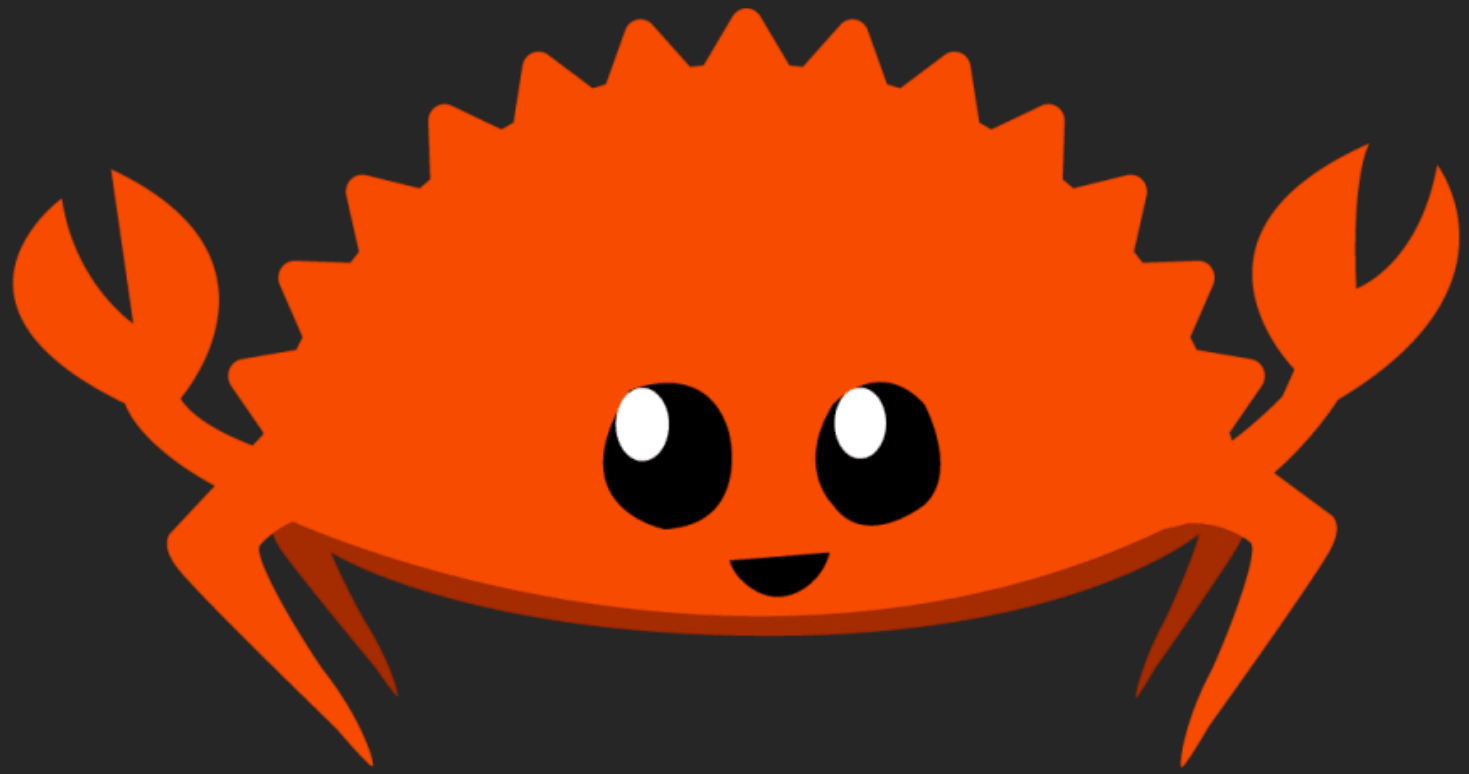
1.5 Sahiplik

```rust
fn main() {
    let mut s1: String = String::from("Hello");

    let r1: &mut String = &mut s1;

    // cannot borrow `s` as mutable more than once at a time.
    // Rust gives this error to prevent data races at compile time.
    // A data race* occurs when there are 2 pointers to the same data
    // and when there is no mechanism to synchronize
    // data access between 2 pointers.
    let r2: &mut String = &mut s1;

    // To fix this error, immutable variable can be used.

    println!("{}, {}", r1, r2);

    let mut s2: String = String::from("World");

    let r3: &String = &s2;

    // cannot borrow `s2` as mutable because
    // it is also borrowed as immutable
    // It is not possible to have a mutable reference
    // if there is an immutable reference already exists.
    let r4: &mut String = &mut s2;

    println!("{}, {}", r3, r4);

    // There is not problem if there are more than 1 immutable references.
    // However, opposite is not possible.

    // * two or more threads in a single process access
    //   the same memory location concurrently, and
    //   at least one of the accesses is for writing, and
    //   the threads are not using any exclusive locks to control
    //   their accesses to that memory.

}       You, 4 days ago • ownership #2
```

```rust
fn main() {
    let mut s: String = String::from("hello world");
    let s2: &str = "hello world";

    let word: usize = first_word(&s);
    let word2: &str = first_word2(s2);

    let hello: &str = &s[..5];

    // Because hello variable sliced 's' string.
    let world: &str = &s[..];

    println!("s: {}, s2: {}, word: {}, word2: {}, hello: {}, world: {}",
    s, s2, word, word2, hello, world);

    s.clear();      You, 4 days ago • slices
}

fn first_word(s: &String) -> usize{
    let bytes: &[u8] = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' '{
            return i;
        }
    }
    s.len()
}

fn first_word2(s: &str) -> &str{
    let bytes: &[u8] = s.as_bytes();

    for (i, &item) in bytes.iter().enumerate() {
        if item == b' '{
            return &s[..i];
        }
    }
    &s[..]
}
```

1.6 Sahiplik - Slicing

```rust
fn main() {
    let a = [1, 2, 3, 4, 5];
    let slice = &a[0..2];

    for i in slice.iter(){
        println!("{}", i);
    }
}
```



Beni Dinlediğiniz İçin Teşekkür Ederim

Yazdığım kodları şu adresten görüntüleyebilirsiniz: https://github.com/MANAPOrkun/Beginner-Rust