Rust'a Giriş

Hafta -1

Kurulum

https://www.rust-lang.org/tools/install

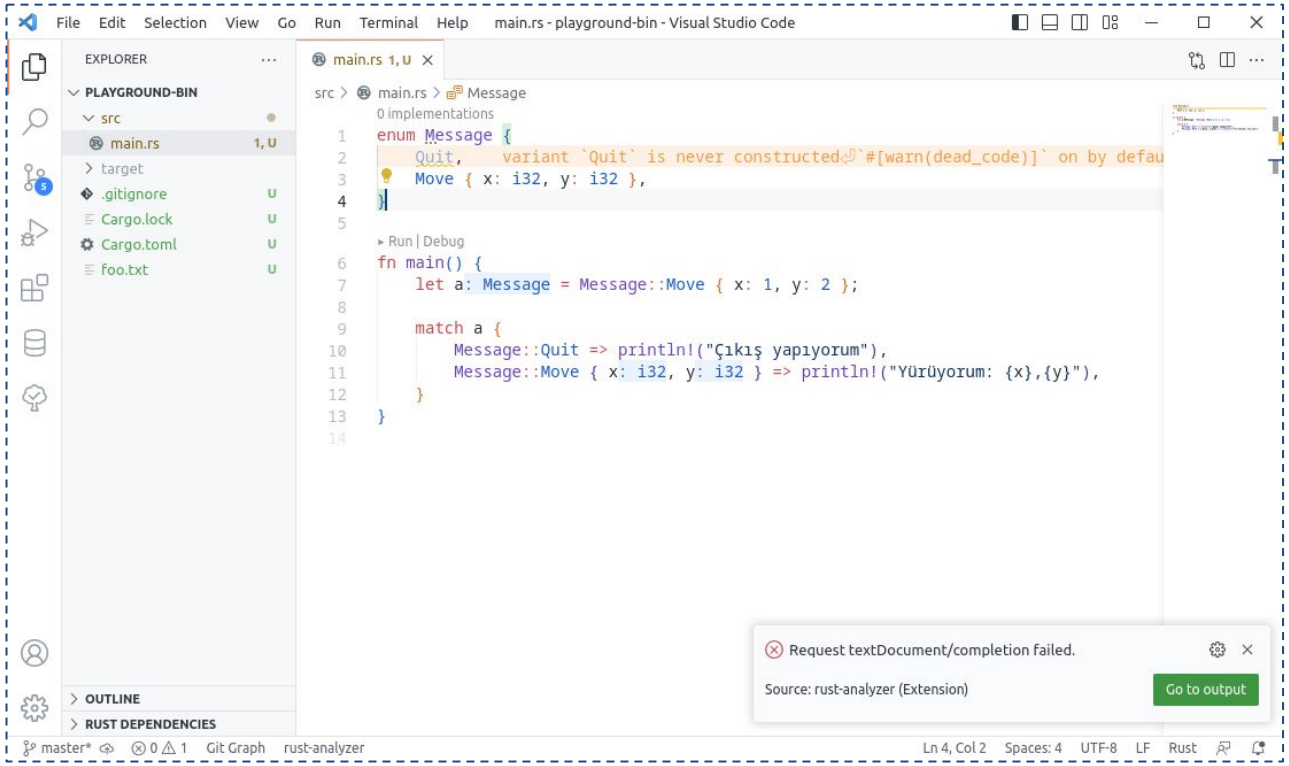
Önerilen IDE: Visual Studio Code

Önerilen VSCode **eklentileri**:

1. rust-analyzer
2. crates
3. Error Lens
4. Better TOML

Önerilen settings.json ayarları:

```
"rust-analyzer.check.command" : "clippy",
"rust-analyzer.checkOnSave" : true,
"[rust]": {
    "editor.defaultFormatter" :
"rust-lang.rust-analyzer" ,
    "editor.formatOnSave" : true
}
```



cargo

Ne için?

- Yeni **proje oluşturmak** (cargo new, cargo init)
- **Derlemek** (cargo build)
- **Bağımlılık(crate) eklemek** (cargo add crateismi)
- **Çalıştırma** (cargo run)
- **Test koşturma** (cargo test)
- **Bağımlı paket lisanslarını kontrol etmek** (cargo deny)
- **Kod formatlama ve linting** (cargo fmt, cargo clippy)
- ...

- Kısaca: Her şey için "cargo"

Yeni uygulama projesi *(varolan dizinde ve dizin ismiyle)*

```
$ cargo init
```

Yeni uygulama projesi

```
$ cargo new uygulamam
```

Yeni kütüphane projesi

```
$ cargo new --lib kutuphanem
```

Derleme

```
$ cargo build
```

Koşturma

```
$ cargo run
```

Testleri koşturma

```
$ cargo test
```

Bağımlılık Ekleme

```
$ cargo add rustfft
```

Bağımlılık Ekleme (Cargo.toml)

```
[dependencies]
paket = "versiyon"

rustfft = "6.0.1"
```

Bir Rust Projesinin Hiyerarşisi

```
-- Proje bilgileri /Cargo.toml
-- Kaynak kodlar /src
-- Uygulamanın ana kaynak kodu /main.rs
-- (crate ise, kütüphanenin ana kaynak kodu) /lib.rs
-- Derlenmiş dosyalar (.exe, .so, .dylib …) /target
```

Syntax

Tam Sayı Değişkenleri

```
let a = 1;

let b = 1i32;

let c:i32 = 1;
```

Boyutlar

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

Floating Değişkenler (virgüllü sayılar)

```
let pi:f32 = 3.14159;

let pi:f64 = 3.14159;
```

Boyutlar

Length	Type
32-bit	f32
64-bit	f64

Boolean Değişkenler (doğru/yanlış)

```
let a:bool = true;

let b = false;
```

Boyutlar

Length	Type
8-bit	bool

Karakter Değişkenleri

```
let karakter:char = 'z';
let garip_z = 'ℤ';
let emoji = '😻';
```

Boyutlar

```
Length          Type
32-bit          char        (4 bytes Unicode Scalar Value)
```

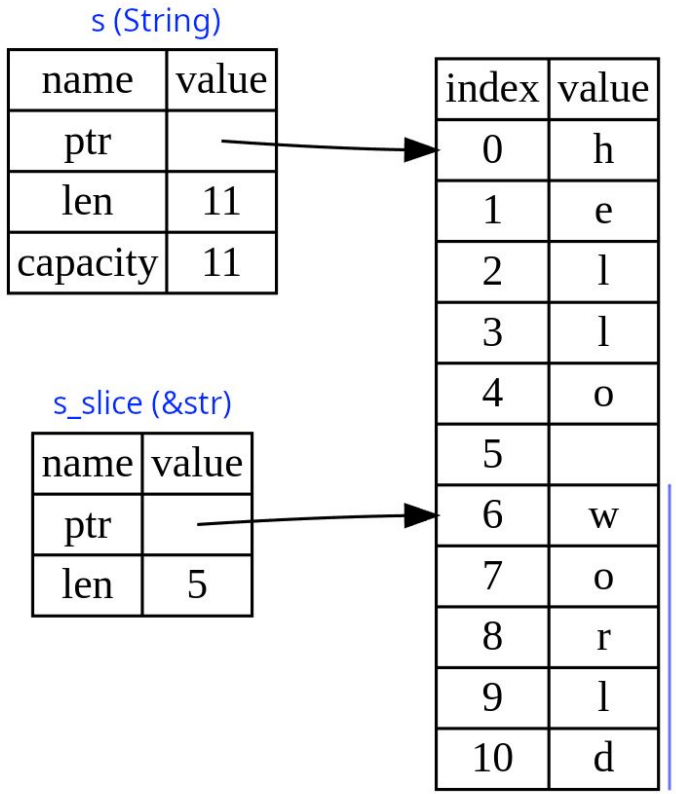
Metin Değişkenleri

```
let s:&str = "Bir metin";

let s:String = String::from("Hello World");
let s_slice:&str = &s[6..11]
```

`&str`: UTF-8 karakter dizisi referansı(string slice). Bellekte bulunan metnin sahibi değil. Dolayısıyla büyüyemez, küçülemez, yeni karakter eklenemez, içeriğindeki karakterleri değiştirebilir fakat sadece bellekteki belirli bölgedeki metni **gösterir**. (C++'taki std::string_view veya C'deki char* gibi fakat daha yetenekli)

`String`: Heap'te oluşturulan, içinde tuttuğu metnin sahibi, mutable(değiştirilebilir) dinamik bir struct. İçerisinde tuttuğu UTF-8 karakter dizisinin sahibi olduğu için düzenleyebilir, silebilir, boyutunu değiştrebilir, .push() metoduyla bellekten yeni bytelar isteyerek üzerine yeni metinler ekleyebilir. (c++'taki std::string gibi)



Değişken Atamaları

```
let i = 1;

i = 5;
```

Terminal Çıktısı

```
error[E0384]: cannot assign twice to immutable variable `i`
  --> main.rs:14:5
   |
13 |     let i = 1;
   |         -
   |         |
   |         first assignment to `i`
   |         help: consider making this binding mutable: `mut i`
14 |     i = 5;
   |     ^^^^^ cannot assign twice to immutable variable
```

Mutability (değiştirilebilirlik)

```
let mut i = 1;

i = 5; // Ok!
```

Shadowing (Gölgede bırakma)

```
let i = 1;

let i = 5; // artık i, 5'i tutan değişkeni gösteriyor
```

```
{
  let i = 1;

  {
    let i = 5;
    // bu scope içinde i = 5

  }
  // bu scope içinde i = 1

}
```

Fonksiyonlar

```
fn main() {

}
```

Değer Döndüren Fonksiyonlar

```
fn main() -> i32 {
    0
    // veya return 0;
}
```

Fonksiyonlar

```
fn karesi( sayi:i32 ) -> i32 {
    return sayi * sayi;
}

fn main() {
     karesi(2); // 4
}
```

Fonksiyonlar

```
fn karesi( sayi:i32 ) -> i32 {
    sayi * sayi // idiomatic return
}

fn main() {
     karesi(2); // 4
}
```

Birleşik Değişkenler - Tuple

```
let tup:(i32, f64, u8) = (500, 6.4, 1);
let xyz = (1.0, 2.0, 3.0);

// Tek Satırda Ayrıştırma
let (x, y, z) = xyz;

// Teker Teker Ayrıştırma
let x = xyz.0;
let y = xyz.1;
let z = xyz.2;
```

Birleşik Değişkenler - Struct

```
struct User {
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64,
}

fn main() {
    // Yeni bir nesne oluşturma
    let user = User {
        active: true,
        username: String::from("kullanici"),
        email: String::from("kullanici@mail.com"),
        sign_in_count: 0,
    };

}
```

Birleşik Değişkenler - Struct

```
struct User { name:String }

impl User {
    fn greet(&self) -> String {
        format!("Merhaba ben {}", self.name)
    }
}

fn main() {
    let user = User {
        name: String::from("Emin"),
    };

    println!("{}", user.greet());
}
```

Terminal Çıktısı

```
Merhaba ben Emin
```

Enum

```
enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
    ChangeColor(i32, i32, i32),
}

fn main() {
    // Veri tipini “Message” diye açıkça belirterek:
    let msg_quit:Message = Message::Quit;
    let msg_move:Message = Message::Move { x: 1, y: 2 };

    // Verinin tipinin “Message” olduğunu derleyici anlar:
    let msg_write = Message::Write(String::from("Writing"));
    let msg_color = Message::ChangeColor(255, 0, 0);
}
```

Enum

```
enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
    ChangeColor(i32, i32, i32),
}

fn print_enum(msg: &Message) {
    match msg {
        Message::Quit => println!("I quit!"),
        Message::Move { x, y } => println!("{}, {}", x, y),
        Message::Write(str) => println!("{}", str),
        Message::ChangeColor(r, g, b) => println!("{r}, {g}, {b}"),
    }
}

fn main() {
    let msg_quit = Message::Quit;
    let msg_move = Message::Move { x: 1, y: 2 };
    let msg_write = Message::Write(String::from("Writing"));
    let msg_color = Message::ChangeColor(255, 0, 0);

    print_enum(&msg_quit);
    print_enum(&msg_move);
    print_enum(&msg_write);
    print_enum(&msg_color);
}
```

Terminal Çıktısı

```
I quit!
1, 2
Writing
255, 0, 0
```

Array(Dizi)

```
fn main() {
    let array:[i32; 3] = [1, 2, 3];
}
```

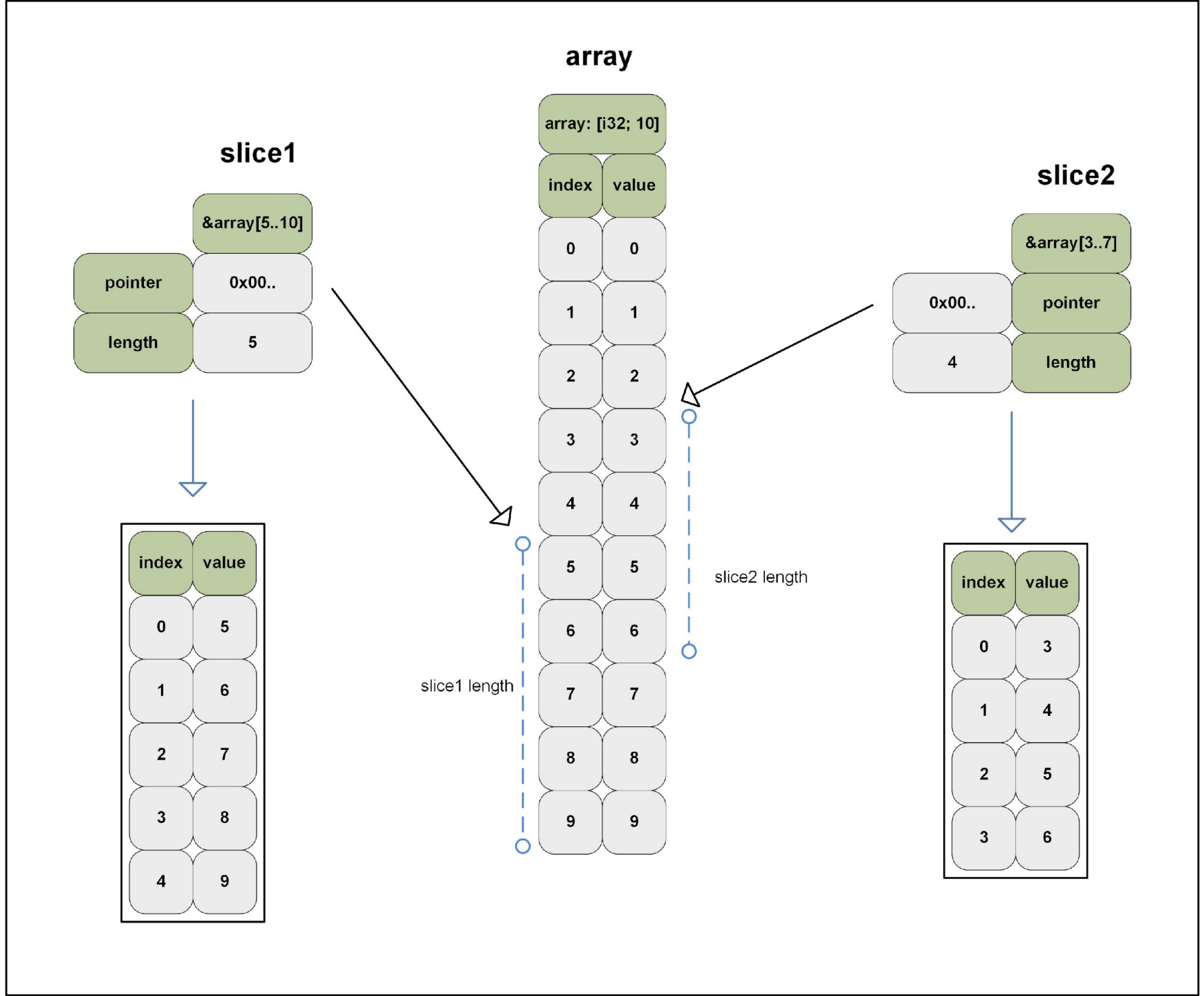
Array(Dizi) ve Slice(Dilim)

```
fn main() {
    // 6 elemanlı i32 tipli bir dizi
    let array: [i32; 6] = [10, 20, 30, 40, 50, 60];
    println!("array: {array:?}");

    // i32 tipli bir dizinin seçilmiş bir aralıktaki dilimi
    let slice: &[i32] = &array[2..4];
    println!("slice: {slice:?}");
}
```

Terminal Çıktısı

```
array: [10, 20, 30, 40, 50, 60]
slice: [30, 40]
```


array
array: [i32; 10]
index
value
0
0
1
1
2
2
3
3
4
4
5
5
6
6
7
7
8
8
9
9
slice1
&array[5..10]
pointer
0x00..
length
5
index
value
0
5
1
6
2
7
3
8
4
9
slice1 length
slice2 length
slice2
&array[3..7]
0x00..
pointer
4
length
index
value
0
3
1
4
2
5
3
6

Array(Dizi)

```
fn main() {
    let array = [1, 2, 3]; // [i32; 3]

    array[3] = 4;  // index out of bounds
    array[0] = 10; // cannot assign immutable variable
}
```

Array(Dizi)

```
fn main() {
    let mut array = [1, 2, 3];

    array[3] = 4;  // index out of bounds
    array[0] = 10; // dizi artık mutable!
}
```

Vec(vector, dinamik dizi)

```
fn main() {
    let mut vector = Vec::from([1, 2, 3]);
}
```

Vec(vector, dinamik dizi)

```
fn main() {
    let mut vector = vec![1, 2, 3]; // vec! bir macro
}
```

Vec(vector, dinamik dizi)

```
fn main() {
    let mut vector = vec![0; 3]; // vector = [0, 0, 0]

    // Program çalışırken ekleme çıkarma yapılabilir.
    vector.push(4); // vector = [0, 0, 0, 4]
    vector.pop();   // vector = [0, 0, 0]
}
```

HashMap(Dictionary)

```
fn main() {
    let mut hm:HashMap<i32, &str> = HashMap::new();

    hm.insert(42, "masa");
    hm.insert(16, "sandalye");

    println!("{:?}", hm);
}
```

Terminal Çıktısı

```
{42: "masa", 12: "sandalye"}
```

HashMap(Dictionary)

```
fn main() {
    let mut hm = HashMap::new(); // tipi explicit yazmadık çünkü

    hm.insert(42, "masa");      // ileride kullanılan veri tipinden
                                // derleyici, HashMap'in tipini algılar
    hm.insert(16, "sandalye");

    println!("{:?}", hm);
}
```

Terminal Çıktısı

```
{42: "masa", 12: "sandalye"}
```

Ownership

Değişkenin sahibi yalnızca belirli bir scope'tur.

```
fn use_hashmap(hm: HashMap<i32, &str>) {

}

fn main() {
    let mut hm = HashMap::new();

    hm.insert(42, "masa");
    hm.insert(16, "sandalye");

    use_hashmap(hm); // !!!!

    println!("{:?}", hm);
}
```

Terminal Çıktısı

```
11 |     use_hashmap(hm);
   |                 -- value moved here
12 |
13 |     println!("{:?}", hm);
   |                       ^^ value borrowed here after move
```

Ownership(Sahiplik)

```
fn use_hashmap(hm:HashMap<i32, &str>) -> HashMap<i32, &str> {
     hm
}

fn main() {
    let mut hm = HashMap::new();

    hm.insert(42, "masa");
    hm.insert(16, "sandalye");

    let mut hm = use_hashmap(hm);

    println!("{:?}", hm);
}
```

Terminal Çıktısı

```
{42: "masa", 12: "sandalye"}
```

Borrowing

Değişkenin kendisi yerine adresi ile erişimi başka scope'lardan sağlanabilir

```
fn use_hashmap(hm: &HashMap<i32, &str>) { // Bir struct reference'ı aldık

}

fn main() {
    let mut hm = HashMap::new();

    hm.insert(42, "masa");
    hm.insert(16, "sandalye");

    use_hashmap(&hm); // Sahipliğini vermedi, ödünç verdi, reference

    println!("{:?}", hm);
}
```

Terminal Çıktısı

```
{42: "masa", 12: "sandalye"}
```

Borrowing(Ödünç Verme)

```
struct Nokta { x:i32, y:i32 }

fn main() {
    let mut nokta = Nokta { x: 1, y: 2 };

    // Aynı anda birden fazla immutable referans uygun
    let referans1 = &nokta;
    let referans2 = &nokta;

    println!("referans1.x ile referans2.x aynı mı?");
    println!("{} == {}", referans1.x, referans2.x);
}
```

Terminal Çıktısı

```
referans1.x ile referans2.x aynı mı?
1 == 1
```

Borrowing(Ödünç Verme)

```rust
struct Nokta { x:i32, y:i32 }

fn main() {
    let mut nokta = Nokta { x: 1, y: 2 };

    // Aynı anda birden fazla mutable referans OLMAZ
    let referans1 = &mut nokta;
    let referans2 = &mut nokta;

    println!("referans1.x ile referans2.x aynı mı?");
    println!("{} == {}", referans1.x, referans2.x);
}
```

Terminal Çıktısı

```
error[E0499]: cannot borrow `nokta` as mutable more than once at a time
  --> src/main.rs:10:21
   |
9  |     let referans1 = &mut nokta;
   |                     ---------- first mutable borrow occurs here
10 |     let referans2 = &mut nokta;
   |                     ^^^^^^^^^^ second mutable borrow occurs here
...
13 |     println!("{} == {}", referans1.x, referans2.x);
   |                          ----------- first borrow later used here
```

Borrowing(Ödünç Verme)

```
struct Nokta { x:i32, y:i32 }

fn main() {
    let mut nokta = Nokta { x: 1, y: 2 };

    let referans1 = &mut nokta;
    println!("referans1.x = {}", referans1.x); // referans1 bir daha kullanılmadı

    // Bir &mut referans bir daha kullanılmamışsa yeni bir &mut referans alınabilir.
    // Aynı lifetime'a sahip birden fazla &mut kullanılamaz.
    let referans2 = &mut nokta;
    println!("referans2.x = {}", referans2.x);
}
```

Terminal Çıktısı

```
referans1.x = 1
referans2.x = 1
```

Control Flow: if, if let

```
fn main() {
    let a = 1;

    if a == 1 {
        println!("a == 1");
    } else {
        println!("a != 1");
    }
}
```

Terminal Çıktısı

```
a == 1
```

Control Flow: if, if let

```rust
fn main() {
    let a:Option<i32> = Some(1); // Option: dolu/boş olabilen bir enum

    if let Some(deger) = a {
        println!("Bir değer varmış: {}", deger);
    } else {
        println!("a'nın içinde bir değişken yok.");
    }
}
```

Terminal Çıktısı

```
Bir değer varmış: 1
```

Control Flow: if, if let

```
fn main() {
    let a:Option<i32> = None;

    if let Some(deger) = a {
        println!("Bir değer varmış: {}", deger);
    } else {
        println!("a'nın içinde bir değişken yok.");
    }
}
```

Terminal Çıktısı

```
a'nın içinde bir değişken yok.
```

Control Flow: match

```
fn main() {
    let a = 1;

    match a {
        1 => println!("a == 1"),
        _ => println!("a != 1"),
    }
}
```

Terminal Çıktısı

```
a == 1
```

Control Flow: match

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

    match a {
        0..=4  => println!("0 ≤ a ≤ 4"),
        5..=10 => println!("5 ≤ a ≤ 10"),
        _ => (),
    }
}
```

Terminal Çıktısı

```
5 ≤ a ≤ 10
```

Control Flow: match

```
fn main() {
    let a:Option<i32> = None;

    match a {
        Some(deger) => println!("{deger} var"),
        None => println!("Bir şey yok"),
    }
}
```

Terminal Çıktısı

```
Bir şey yok
```

Control Flow: match

```
fn main() {
    let a:Option<i32> = Some(1);

    match a {
        Some(deger) => println!("{deger} var"),
        None => println!("Bir şey yok"),
    }
}
```

Terminal Çıktısı

```
1 var
```

Control Flow: match

```
enum Message {
    Quit,
    Move { x: i32, y: i32 },
}

fn main() {
    let a = Message::Move{ x: 1, y: 2 };

    match a {
        Message::Quit => {
            println!("Çıkış yapıyorum");
        },
        Message::Move {x, y} => println!("Yürüyorum: {x},{y}"),
    }
}
```

Terminal Çıktısı

```
Yürüyorum: 1,2
```

Loops(Döngüler): for

```
fn main() {
      for i in 1..=41 {
            println!("maşaallah");
      }
}
```

Terminal Çıktısı

```
maşaallah
maşaallah
maşaallah
…
```

Loops(Döngüler): while

```
fn main() {
     let mut i = 1;

     while i <= 41 {
          println!("maşaallah");

          i += 1;
     }
}
```

Terminal Çıktısı

```
maşaallah
maşaallah
maşaallah
…
```

Loops(Döngüler): loop

```
fn main() {
     let mut i = 1;

     loop {
          println!("maşaallah");

          i += 1;
          if i > 41 {
               break;
          }
     }
}
```

Terminal Çıktısı

```
maşaallah
maşaallah
maşaallah
…
```

Loops(Döngüler): while let

```
fn main() {
	let mut x = vec![1, 2, 3];

	while let Some(deger) = x.pop() {
		println!("deger = {}", deger);
	}
}
```

Terminal Çıktısı

```
deger = 1
deger = 2
deger = 3
```

Sıra Sizde

1. `fn hello(name: &str) -> String`

   - fonksiyonu “Hello {name}!” metnini döndürecek

2. `fn double_if_even(num: i32) -> i32`

   - fonksiyonu çift sayı verilirse 2 katını, tek sayı verilirse aynısını döndürecek.

3. `fn multiply_pi(num: f32) -> f32`

   - fonksiyonu verilen sayıyı Pi sayısı ile çarpıp geri döndürecek.