

Rust'a Giriş

Hafta -1

Kurulum

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

cargo

Ne için kullanılır?

- Yeni proje oluşturmak
- Projeyi derlemek
- Projeye bağımlılık (paket) eklemek
- Projeyi çalıştırmak
- Projeyi test etmek
-
- Kısaca: Her şey için "cargo"

cargo

Yeni proje

```
$ cargo init
```

cargo

Derleme

```
$ cargo build
```

cargo

Bağımlılık ekleme

Cargo.toml:

```
[dependencies]  
paket = "versiyon"  
  
rustfft = "6.0.1"
```

cargo

Çalıştırma

```
$ cargo run
```

cargo

Testleri çalıştırma

\$ cargo test

cargo

Proje hiyerarşisi

/Cargo.toml	- Proje bilgileri
/src	- Kaynak kodlar
--/main.rs	
--/lib.rs	- *eğer bir crate ise
/target	- Exe çıktısı ve bağımlılıklar

Syntax

Değişkenler

```
let a = 1;  
  
let b = 1i32;  
  
let c:i32 = 1;
```

Syntax

Değişkenler

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

Syntax

Değişkenler

```
let i = 1;
```

```
i = 5;
```

```
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
```

Syntax

Mutability (Değiştirilebilirlik)

```
let mut i = 1;  
  
i = 5; // Ok!
```

Syntax

Fonksiyonlar

```
fn main() {  
  
}
```

Syntax

Fonksiyonlar

```
fn karesi( sayi:i32 ) -> i32 {  
    return sayi * sayi;  
}  
  
fn main() {  
    karesi(2); // 4  
}
```

Syntax

Fonksiyonlar

```
fn karesi( sayi:i32 ) -> i32 {  
    sayi * sayi  
}  
  
fn main() {  
    karesi(2); // 4  
}
```


Syntax

String ve str

```
fn selam( isim:&str ) -> String {  
    format!("Selam {}", isim)  
}  
  
fn main() {  
    let isim = "Emin";  
  
    selam(isim); // "Selam Emin"  
}
```

&str : Derleme zamanında programa gömülen, sadece okunabilen static bir string

String: Heap'te oluşturulan dinamik bir String struct'ı

Syntax

String ve str

```
fn selam( isim:&str ) -> String {  
    format!("Selam {}", isim)  
}  
  
fn main() {  
    let isim = "Emin";  
  
    println!("{}", selam(isim));  
}
```

> Selam Emin

Syntax

Vec ve array

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

Syntax

Vec ve array

```
fn main() {  
    let array = [1, 2, 3];  
  
    array[3] = 4; // index out of bounds  
    array[0] = 10; // cannot assign immutable variable  
}
```

Syntax

Vec ve array

```
fn main() {  
    let mut array = [1, 2, 3];  
  
    array[3] = 4; // index out of bounds  
    array[0] = 10;  
}
```

Syntax

Vec ve array

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

Syntax

Vec ve array

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

Syntax

Vec ve array

```
fn main() {  
    let mut vector = vec![0; 3];  
  
    vector.push(4); // vector = [0, 0, 0, 4]  
    vector.pop();  // vector = [0, 0, 0]  
}
```


Syntax

HashMap (Dictionary)

```
fn main() {  
    let mut hm:HashMap<i32, &str> = HashMap::new();  
  
    hm.insert(42, "masa");  
    hm.insert(16, "sandalye");  
  
    println!("{:?}", hm);  
}
```

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

Syntax

HashMap (Dictionary)

```
fn main() {  
    let mut hm = HashMap::new();  
  
    hm.insert(42, "masa");    // derleyici, HashMap'in tipini algılar  
    hm.insert(16, "sandalye");  
  
    println!("{:?}", hm);  
}
```

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

Ownership

```
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);  
}
```

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

Ownership

```
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);  
}
```

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

Borrowing

```
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); // Sahipliğini almadı, reference  
  
    println!("{:?}", hm);  
}
```

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

Syntax

Control Flow: if, if let

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

```
> a == 1
```

Syntax

Control Flow: if, if let

```
fn main() {  
    let a:Option<i32> = Some(1);  
  
    if let Some(deger) = a {  
        println!("Bir değer varmış: {}", deger);  
    } else {  
        println!("a nullmuş.");  
    }  
}
```

> Bir değer varmış: 1

Syntax

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 nullmuş.");  
    }  
}
```

> a nullmuş.

Syntax

match

```
fn main() {  
    let a = 1;  
    match a {  
        1 => println!("a == 1"),  
        _ => println!("a != 1"),  
    }  
}
```

> a == 1

Syntax

match

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

> 5 ≤ a < 10

Syntax

Loops: for

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

```
> maşaallah  
> maşaallah  
> maşaallah  
> ...
```

Syntax

Loops: while

```
fn main() {  
    let mut i = 1;  
  
    while i <= 41 {  
        println!("maşaallah");  
  
        i += 1;  
    }  
}
```

```
> maşaallah  
> maşaallah  
> maşaallah  
> ...
```

Syntax

Loops: loop

```
fn main() {  
    let mut i = 1;  
  
    loop {  
        println!("maşaallah");  
  
        i += 1;  
        if i > 41 {  
            break;  
        }  
    }  
}
```

```
> maşaallah  
> maşaallah  
> maşaallah  
> ...
```

Syntax

Loops: while let

```
fn main() {  
    let mut x = vec![1, 2, 3];  
  
    while let Some(deger) = x.pop() {  
        println!("deger = {}", deger);  
    }  
}
```

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

Homeworks

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

will return: "Hello {name}!"

2. Write `fn make_double(num:i32) -> i32`

will return: 4 when given 2

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

will multiply the num with $\text{Pi}(\pi)$ and return the result.