

Rust: Enum, Trait, Macro

Hafta -2

Enum

```
enum IpAddr {  
    V4,  
    V6,  
}  
  
fn main() {  
    let v = IpAddr::V4;  
}
```

Enum

```
enum IpAddr {  
    V4(String),  
    V6(String),  
}  
  
fn main() {  
    let v = IpAddr::V4(String::from("127.0.0.1"));  
}
```

Enum

```
enum IpAddr {  
    V4(String),  
    V6(String),  
}  
  
fn main() {  
    let v = IpAddr::V4(String::from("127.0.0.1"));  
  
    let ip_str = match v {  
        IpAddr::V4(s) => s,  
        IpAddr::V6(s) => s,  
    };  
  
    println!("{}", ip_str);  
}
```

> 127.0.0.1

Enum

```
enum IpAddr {  
    V4(String),  
    V6(String),  
}  
  
fn main() {  
    let v = IpAddr::V6(String::from("::1"));  
  
    let ip_str = match v {  
        IpAddr::V4(s) => s,  
        IpAddr::V6(s) => s,  
    };  
  
    println!("{}", ip_str);  
}
```

> ::1

Result

```
pub enum Result<T, E> {  
    Ok(T),  
    Err(E),  
}
```

Result

```
fn main() {  
    let x: Result<i32, &str> = Ok(-3);  
    assert_eq!(x.is_ok(), true);  
  
    let x: Result<i32, &str> = Err("Hata oldu");  
    assert_eq!(x.is_err(), true);  
}
```

Result

```
fn main() {  
    let x: Result<i32, &str> = Ok(-3);  
  
    match x {  
        Ok(i) => assert_eq!(i, -3),  
        Err(s) => println!(s),  
    }  
}
```


Result örnek

```
use std::net::UdpSocket;  
use std::io;  
  
fn main() {  
    let socket = UdpSocket::bind("127.0.0.1:1453");  
}
```

Result örnek

```
use std::net::UdpSocket;
use std::io;

fn main() {
    let socket:Result<UdpSocket, io::Error> = UdpSocket::bind("127.0.0.1:1453");
}
```

Result örnek

```
use std::net::UdpSocket;
use std::io;

fn main() {
    let socket:Result<UdpSocket, io::Error> = UdpSocket::bind("127.0.0.1:1453");

    let socket:UdpSocket = match socket {
        Ok(sock) => sock,
        Err(err) => panic!("Bind etme hatası: {}", err);
    };
}
```

Result örnek

```
use std::net::UdpSocket;
use std::io;

fn main() {
    let socket:Result<UdpSocket, io::Error> = UdpSocket::bind("127.0.0.1:1453");

    // Ok() ise socketi döndürür, Err() ise panic!
    let socket:UdpSocket = socket.unwrap();
}
```

Result örnek

```
use std::net::UdpSocket;
use std::io;

fn main() {
    let socket:Result<UdpSocket, io::Error> = UdpSocket::bind("127.0.0.1:1453");

    let socket:UdpSocket = socket.expect("Bind etme hatası");
}
```

Result Kolay Syntax

```
use std::net::UdpSocket;
use std::io;

fn main() -> io::Result<()> {
    let socket = UdpSocket::bind("127.0.0.1:1453"?);

    Ok(())
}
```

io::Result'un tanımı:

```
pub type io::Result<T> = Result<T, io::Error>;
```

Soru işaretini(?) **Result** döndüren fonksiyonların içinde kısaltma olarak kullanabilirsiniz. ? ile değeri elde edilen **Result**, eğer **Err** değerindeyse, soru işareti(?) bulunduğu fonksiyonu **return Err(...)** çalıştırarak sonlandırır.

Dolayısıyla **Err**, bir üst fonksiyona yollanmış olur.

Result Kolay Syntax

```
use std::net::UdpSocket;
use std::io;

fn main() -> io::Result<()> {
    let socket = UdpSocket::bind("127.0.0.1:1453")?;

    let mut buf = [0; 1024];
    let (len, sender) = socket.recv_from(&mut buf).expect("recv_from hatası");

    Ok(())
}
```

Result Kolay Syntax

```
use std::net::UdpSocket;
use std::io;

fn main() -> io::Result<()> {
    let socket = UdpSocket::bind("127.0.0.1:1453")?;

    let mut buf = [0; 1024];
    let (len, sender) = socket.recv_from(&mut buf)?;

    Ok(())
}
```


Option

```
pub enum Option<T> {  
    None,  
    Some(T),  
}
```

Null yani **None** olabilir değerleri tutmak için kullanılan bir enum.

Option

```
fn main() {  
    let x: Option<i32> = Some(-3);  
    assert_eq!(x.unwrap(), -3);  
  
    let x: Option<i32> = None;  
    assert_eq!(x.unwrap(), -3); // panic!  
}
```

Option

```
fn main() {  
    let x: Option<i32> = Some(-3);  
    match x {  
        Some(i) => assert_eq!(i, -3),  
        None => println!("Deger yok."),  
    }  
}
```

Option

```
fn main() {  
    let x: Option<i32> = Some(-3);  
    if let Some(i) = x {  
        assert_eq!(i, -3);  
    }  
}
```

Struct

```
struct Point {  
    x: i64,  
    y: i64  
}
```

Struct

```
struct Point {  
    x: i64,  
    y: i64  
}  
  
struct TupleOrnek(i32, i32);
```

Struct

```
struct Point {  
    x: i64,  
    y: i64  
}  
  
struct TupleOrnek(i32, i32);  
  
struct VeriYok;
```

Struct

```
struct Point {  
    x: i64,  
    y: i64  
}  
  
fn main() {  
    let p = Point { x:1, y:2 };  
  
    let px = p.x;  
    let py = p.y;  
}
```


Struct

```
struct Point {  
    x: i64,  
    y: i64  
}  
  
fn main() {  
    let p = Point { x:1, y:2 };  
  
    let Point {x, y} = p;  
  
    x; // kullanılabilir  
    y;  
}
```

Struct

```
struct Point(i64, i64);  
  
fn main() {  
    let p = Point(1, 2);  
  
    let px = p.0;  
    let py = p.1;  
}
```

Struct

```
struct Point(i64, i64);

fn main() {
    let p = Point(1, 2);

    let (px, py) = p;

    px; // kullanılabilir
    py;
}
```

Struct

```
struct Person {  
    name: String,  
    age: u8  
}  
  
fn main() {  
    let p = Person { name: String::from("Emin"), age:24 };  
}
```

Struct

```
#[derive(Debug)]
struct Person {
    name: String,
    age: u8
}

impl Person {
    fn yas_kac(&self) -> u8 { self.age }
    fn yas_ayarla(&mut self, yas: u8) { self.age = yas; }

    fn new(name: String, age: u8) -> Self {
        Self {name, age}
    }
}

fn main() {
    let mut p = Person::new(String::from("Emin"), 24);

    p.yas_ayarla(27);

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

Trait

```
trait Speaker {  
    fn say_name(&self) -> String;  
    fn say_age(&self) -> String;  
}
```

Trait'ler diğer dillerdeki **interface**'lerdir.

Adından da anlaşılacağı üzere, bir davranış biçimi taslağı sunar.

Örnek: "bir konuşmacı olabilmek için iki metodu implement etmeli."

Trait

```
trait Speaker {  
    fn say_name(&self) -> String;  
    fn say_age(&self) -> String;  
}  
  
struct Person {  
    name: String,  
    age: u8  
}
```

Trait

```
trait Speaker {  
    fn say_name(&self) -> String;  
    fn say_age(&self) -> String;  
}  
  
struct Person {  
    name: String,  
    age: u8  
}  
  
impl Speaker for Person {  
    fn say_name(&self) -> String {  
        format!("My name is {}", self.name)  
    }  
    fn say_age(&self) -> String {  
        format!("My age is {}", self.age)  
    }  
}
```


Trait

```
trait Speaker {  
    fn say_name(&self) -> String;  
    fn say_age(&self) -> String;  
}  
  
struct Person { name: String, age: u8 }  
  
impl Speaker for Person {  
    fn say_name(&self) -> String { format!("My name is {}", self.name) }  
    fn say_age(&self) -> String { format!("My age is {}", self.age) }  
}  
  
fn say_your_name(speaker: &dyn Speaker) {  
    println!("{}", speaker.say_name());  
}  
  
fn main() {  
    let s = Person { name: String::from("Emin"), age: 24 };  
  
    say_your_name(&s);  
}
```

> My name is Emin

Trait

```
fn say_your_name(speaker: &dyn Speaker) {  
    println!("{}", speaker.say_name());  
}  
  
// veya generic tip kullanarak:  
  
fn say_your_name<T: Speaker>(speaker: &T) {  
    println!("{}", speaker.say_name());  
}
```

Trait örnek: Display

Trait `std::fmt::Display` 

1.0.0 [-][src]

```
pub trait Display {  
    fn fmt(&self, f: &mut Formatter<'_>) -> Result<(), Error>;  
}
```

Trait

```
use std::fmt;

struct Person { name: String, age: u8 }

impl fmt::Display for Person {
    fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
        write!(f, "{}({})", self.name, self.age)
    }
}

fn main() {
    let s = Person { name: String::from("Emin"), age: 24 };

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

> Emin(24)

Macro

```
// Function-like macros:  
println!("Selam");  
  
let v = vec![1, 2, 3];  
  
if cfg!(target_os = "linux") {  
    // eger linux ise ...  
}
```

Macro

```
// Attribute macros:  
#[cfg(target_os = "linux")]  
fn sadece_linuxta() {}  
  
#[test]  
fn test_fonksiyonudur() {}  
  
#[async_std::main]           // async-std  
async fn main() {}  
  
#[get("/")]                   // rocket  
fn index() -> &'static str {  
    "Hello, world!"  
}
```

Macro


```
macro_rules! merhaba {  
    () => {  
        println!("Merhaba!")  
    };  
}  
  
fn main() {  
    merhaba();  
}
```



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

Macro

```
macro_rules! dizi {  
  [] => (                                // vec![]  
    Vec::new()  
  );  
  [$( $x:expr ),+ $( , )? ] => ( // vec![1, 2, 3]  
  {  
    let mut v = Vec::new();  
  
    $( v.push($x); )*  
  
    v  
  }  
);  
}  
  
fn main() {  
  let mut v = dizi![];  
}
```



```
fn main() {  
  let mut v = Vec::new();  
}
```


Macro

```
macro_rules! dizi {  
  [] => (                                // vec![]  
    Vec::new()  
  );  
  [$( $x:expr ),+ $( , )? ] => ( // vec![1, 2, 3]  
    {  
      let mut v = Vec::new();  
  
      $( v.push($x); )*  
  
      v  
    }  
  );  
}  
  
fn main() {  
  let mut v = dizi![1, 2, 3];  
}
```



```
fn main() {  
  let mut v = {  
    Vec::new();  
  
    v.push(1);  
    v.push(2);  
    v.push(3);  
  
    v  
  }  
}
```

Homeworks

1. Write `fn to_letter_grade(num:u8) -> String`
2. Write `fn log(level:LogLevel, msg:&str) -> String`
 - LogLevel is an enum. Prints logs with level tag: [WARN]: This is warning log.
3. Write `Person {name:String, age: u8, gender: Gender}`
 - Gender is enum. Implement Display trait for Person.
4. Write `display!(Person, "{}({})", name, gender)`
 - This macro will implement Display trait for Person with format you provide.