Referensi Cepat Rust

Ownership, tipe, trait, pattern matching esensial

Dasar

Hello World

```
fn main() {
    println!("Hello, World!");
}
```

Perintah Cargo

```
cargo new my_project    # buat proyek baru
cargo build             # kompilasi (debug)
cargo build --release   # kompilasi (teroptimasi)
cargo run               # build dan jalankan
cargo test              # jalankan tes
```

Struktur Proyek

`Cargo.toml`	Manifest proyek (dependensi, metadata)
`src/main.rs`	Titik masuk binary crate
`src/lib.rs`	Root library crate
`tests/`	Direktori tes integrasi

Variabel & Mutabilitas

Binding & Mutabilitas

```
let x = 5;              // immutable secara default
let mut y = 10;         // mutable
y += 1;
const MAX: u32 = 100;   // konstanta waktu kompilasi
```

Shadowing

```
let x = 5;
let x = x + 1;          // shadows x sebelumnya
let x = "kini string"; // dapat mengubah tipe
```

Tipe Scalar

`i8..i128, isize`	Integer bertanda
`u8..u128, usize`	Integer tidak bertanda
`f32, f64`	Floating point (f64 default)
`bool`	`true` / `false`
`char`	Unicode scalar value (4 byte)

Tipe Compound

```
let tup: (i32, f64, char) = (42, 6.4, 'z');
let (a, b, c) = tup;        // destructure
let arr: [i32; 3] = [1, 2, 3];
let first = arr[0];
```

Fungsi

Definisi

```
fn add(a: i32, b: i32) -> i32 {
    a + b   // tanpa titik koma = ekspresi return
}
```

Closure

```
let double = |x: i32| x * 2;
let sum: i32 = vec![1, 2, 3]
    .iter()
    .map(|x| x * 2)
    .sum();
```

Pointer Fungsi & Trait

`fn(T) -> U`	Tipe pointer fungsi
`Fn(T) -> U`	Closure yang meminjam
`FnMut(T) -> U`	Closure yang meminjam secara mutable
`FnOnce(T) -> U`	Closure yang mengambil ownership

Alur Kontrol

If / Else

```
let status = if score >= 90 { "A" }
             else if score >= 80 { "B" }
             else { "C" };  // if adalah ekspresi
```

Loop

```
loop { break; }                 // tak terbatas
while condition { }             // while
for item in &vec { }            // iterator
for i in 0..10 { }             // range
for (i, v) in vec.iter().enumerate() { }
```

Label Loop

```
'outer: for i in 0..5 {
    for j in 0..5 {
        if i + j > 6 { break 'outer; }
    }
}
```

Ownership & Borrowing

Aturan Ownership

1. Setiap nilai memiliki tepat satu owner.
2. Ketika owner keluar dari scope, nilai di-drop.
3. Nilai dapat dipindahkan atau di-clone.

Move & Clone

```
let s1 = String::from("hello");
let s2 = s1;          // s1 dipindahkan, tidak valid lagi
let s3 = s2.clone();  // deep copy, keduanya valid
```

Borrowing

```
fn len(s: &String) -> usize { s.len() }  // shared ref
fn push(s: &mut String) { s.push('!'); }  // mutable ref
// Aturan: banyak &T ATAU satu &mut T, tidak boleh keduanya
```

Lifetime

```
fn longest<'a>(a: &'a str, b: &'a str) -> &'a str {
    if a.len() > b.len() { a } else { b }
}
```

Struct & Enum

Struct

```
struct User {
    name: String,
    age: u32,
    active: bool,
}
let u = User { name: String::from("Alice"), age: 30, active: true };
```

Blok Impl

```
impl User {
    fn new(name: &str, age: u32) -> Self {
        Self { name: name.to_string(), age, active: true }
    }
    fn greeting(&self) -> String {
        format!("Hi, {}", self.name)
    }
}
```

Enum

```
enum Shape {
    Circle(f64),
    Rect { w: f64, h: f64 },
    Point,
}
let s = Shape::Circle(5.0);
```

Pattern Matching

Ekspresi Match

```
match shape {
    Shape::Circle(r) => std::f64::consts::PI * r * r,
    Shape::Rect { w, h } => w * h,
    Shape::Point => 0.0,
}
```

If Let & While Let

```
if let Some(val) = optional {
    println!("{val}");
}
while let Some(top) = stack.pop() {
    println!("{top}");
}
```

Sintaks Pola

`_`	Wildcard, cocok dengan apa saja
`x @ 1..=5`	Bind range yang cocok ke `x`
`(a, b, ..)`	Destructure tuple, abaikan sisanya
`Some(x) if x > 0`	Match guard
`Foo { x, .. }`	Struct, abaikan field lainnya

Penanganan Error

Result & Option

```
enum Result<T, E> { Ok(T), Err(E) }
enum Option<T>    { Some(T), None }
```

Operator ?

```
fn read_file(path: &str) -> Result<String, io::Error> {
    let mut s = String::new();
    File::open(path)?.read_to_string(&mut s)?;
    Ok(s)
}
```

Menangani Error

```
match result {
    Ok(val) => println!("{val}"),
    Err(e) => eprintln!("Error: {e}"),
}
let val = result.unwrap_or(0);
let val = result.unwrap_or_else(|_| default());
```

Metode Umum

`.unwrap()`	Ambil nilai atau panic
`.expect(msg)`	Ambil nilai atau panic dengan pesan
`.unwrap_or(default)`	Ambil nilai atau gunakan default
`.map(f)`	Transformasi nilai Ok/Some
`.and_then(f)`	Rantai operasi (flatmap)
`.is_ok() / .is_some()`	Pemeriksaan boolean

Referensi Cepat Rust

Trait

Mendefinisikan & Mengimplementasikan

```
trait Summary {
    fn summarize(&self) -> String;
    fn preview(&self) -> String {   // impl default
        format!("{}...", &self.summarize()[..20])
    }
}
impl Summary for User {
    fn summarize(&self) -> String { self.name.clone() }
}
```

Trait Bounds

```
fn notify(item: &impl Summary) { }
fn notify<T: Summary + Display>(item: &T) { }
fn notify(item: &(impl Summary + Display)) { }
```

Trait Umum

`Display`	Pemformatan string untuk pengguna
`Debug`	Pemformatan debug (**`{:?}`**)
`Clone, Copy`	Duplikasi (deep / bitwise)
`PartialEq, Eq`	Perbandingan kesamaan
`PartialOrd, Ord`	Perbandingan urutan
`Iterator`	**`next()`** untuk iterasi
`From, Into`	Konversi tipe
`Default`	Konstruktor nilai default

Koleksi

Vec

```
let mut v: Vec<i32> = vec![1, 2, 3];
v.push(4);
v.pop();              // mengembalikan Option<i32>
let first = &v[0];   // panic jika kosong
let first = v.get(0); // mengembalikan Option<&i32>
```

HashMap

```
use std::collections::HashMap;
let mut m = HashMap::new();
m.insert("key", 42);
m.entry("key").or_insert(0);
if let Some(val) = m.get("key") { }
```

String

```
let s = String::from("hello");
let s = "hello".to_string();
let combined = format!("{} {}", s, "world");
for c in s.chars() { }   // iterasi karakter
```

Iterator

```
let sum: i32 = vec![1, 2, 3].iter().sum();
let doubled: Vec<_> = v.iter().map(|x| x * 2).collect();
let evens: Vec<_> = v.iter().filter(|x| *x % 2 == 0).collect();
```

Konkurensi

Thread

```
use std::thread;
let handle = thread::spawn(|| {
    println!("dari thread yang dibuat");
});
handle.join().unwrap();
```

Channel

```
use std::sync::mpsc;
let (tx, rx) = mpsc::channel();
tx.send(42).unwrap();
let val = rx.recv().unwrap();
```

Shared State

`Arc<T>`	Atomic reference counting (Rc aman untuk thread)
`Mutex<T>`	Mutual exclusion, lock untuk akses nilai dalam
`RwLock<T>`	Banyak reader atau satu writer
`Send`	Trait: aman dipindahkan antar thread
`Sync`	Trait: aman berbagi referensi antar thread

Macro & Atribut

Macro Umum

`println!()`	Cetak dengan baris baru
`format!()`	Kembalikan String yang diformat
`vec![]`	Buat Vec dari literal
`todo!()`	Placeholder, panic saat runtime
`assert!(expr)`	Panic jika expr false
`assert_eq!(a, b)`	Panic jika a != b

Atribut Derive

```
#[derive(Debug, Clone, PartialEq)]
struct Point { x: f64, y: f64 }
// Auto-implementasikan Debug, Clone, PartialEq
```

Atribut Testing

```
#[cfg(test)]
mod tests {
    use super::*;
    #[test]
    fn it_works() { assert_eq!(add(2, 2), 4); }
    #[test]
    #[should_panic]
    fn it_panics() { panic!("boom"); }
}
```