kintsugi-stack-rust

> "Rust is technology from the past come to save the future from itself." — Graydon Hoare, Creator of Rust

- Author: Kintsugi-Programmer

> Disclaimer: The content presented here is a curated blend of my personal learning journey, experiences, open-source documentation, and invaluable knowledge gained from diverse sources. I do not claim sole ownership over all the material; this is a community-driven effort to learn, share, and grow together.

Table of Contents

- kintsugi-stack-rust
  - Table of Contents
- Rust's Official Documentation's Codes
  - 1. Getting Started
    - 1.1. Installation
    - 1.2. Hello, World!
    - 1.3. Cargo
  - 2. Programming a Guessing Game
  - 3. Common Programming Concepts
    - 3.1. Variables and Mutability
    - 3.2. Data Types
    - 3.3. Functions
    - 3.4. Comments
    - 3.5. Control Flow

Rust's Official Documentation's Codes

- parent doc: https://doc.rust-lang.org/stable/
- "the book"(The Rust Programming Language): https://doc.rust-lang.org/stable/book/
- The Rust Standard Library: https://doc.rust-lang.org/stable/std/

1. Getting Started

1.1. Installation

```
sudo apt update
sudo apt install -y curl
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- Rust Lang: Rust Install https://doc.rust-lang.org/book/ch01-01-installation.html
  - Windows
    - Install Linux, Just kidding !!
    - https://visualstudio.microsoft.com/downloads/
      - Install VS
      - Install VSC
      - Install Build Tools for Visual Studio
        - then Restart Computer
    - https://rust-lang.org/tools/install/
      - Install Rust
    - `rustup toolchain install stable-x86_64-pc-windows-gnu`
    - `rustup default stable-x86_64-pc-windows-gnu`
  - Linux: `$ curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh`
- Rust Server Dev: Rust Analyzer Install https://marketplace.visualstudio.com/items?itemName=rust-lang.rust-analyzer

1.2. Hello, World!

```rust
// 1_2_hello_world.rs
fn main(){
    println!("Hello, World! ")
}
// rustc main.rs && ./main
```

- rust code file extension `.rs`

```rust
// 1_2_hello_world.rs
fn main(){
    println!("Hello, World! ")
}
// rustc main.rs && ./main
```

- Compile command

```
rustc main.rs
```

- Rust Binary Run command

```
./main
```

1.3. Cargo

```
cargo --version # cargo version check
&& cargo new project_name # create proj.
&& cd project_name
&& cargo build # build executable
&& cargo run # run project
&& cargo check # check for err without any executable
&& cargo help # help

# .
# ├── Cargo.lock
# ├── Cargo.toml
# └── src
#     └── main.rs

# .
# ├── Cargo.lock
# ├── Cargo.toml
# ├── src
# │   └── main.rs
# └── target
#     ├── CACHEDIR.TAG
#     └── debug
#         ├── build
#         ├── deps
#         │   ├── libone_three_hello_cargo-f9884884092cd48a.rmeta
#         │   ├── one_three_hello_cargo-5885dd703046e3fc
#         │   ├── one_three_hello_cargo-5885dd703046e3fc.d
#         │   └── one_three_hello_cargo-f9884884092cd48a.d
#         ├── examples
#         ├── incremental
```

```
#          |    ├── one_three_hello_cargo-10ah7hvrv4gzi
#          |    |    ├── s-heho9i1rut-1ysdico-bwbrfy6ptxbomf5iwqz5vt3f0
#          |    |    |    ├── dep-graph.bin
#          |    |    |    ├── query-cache.bin
#          |    |    |    └── work-products.bin
#          |    |    └── s-heho9i1rut-1ysdico.lock
#          |    └── one_three_hello_cargo-3dgwin0zbxstr
#          |         ├── s-heho98ij63-039vdoh-143bk3qfw5zxnyx5otl9s0tja
#          |         |    ├── 00ylhni9avwle6wyqpyzm6par.o
#          |         |    ├── 19k6gm7hj98zo0jv2b5mu1std.o
#          |         |    ├── 1jqdhkz0e02p777bbobcmna2j.o
#          |         |    ├── 5fgvfmdk1vtvncsc4ze5a0wi9.o
#          |         |    ├── 8z1o97dthkm4wl9qy6anckmmy.o
#          |         |    ├── 9fwica1fdmiqw5oux5l4cedjc.o
#          |         |    ├── dep-graph.bin
#          |         |    ├── query-cache.bin
#          |         |    └── work-products.bin
#          |         └── s-heho98ij63-039vdoh.lock
#          ├── one_three_hello_cargo
#          └── one_three_hello_cargo.d
```

- Cargo:
  - Rust's Build System
  - - Package manager
  - - Builtin When we Install Rust (Painpoint of other prog. lang.)
- Compile command

```
rustc main.rs
```

- Rust Binary Run command

```
./main
```

- Cargo version check

```
cargo --version
```

- Create New Cargo Project

```
cargo new one_three_hello_cargo
```

- File Organisation

    - `Cargo.toml`
        - package config file
    - `.gitignore`
        - default code ver. ignore file
        - ignore flags for git ver.
    - `\src`
        - contains actual code
        - `main.rs`
            - Starter code

```
.
├── Cargo.lock
├── Cargo.toml
└── src
    └── main.rs
```

- Build command
    - Build
    -
        - Create `Cargo.lock`
        - contain dependencies
    -
        - Create `\target`
        - contain `\debug`
            - contain our actual executable
            -
                - other supporting stuff

```
cargo build
```

- run command

```
cargo run
```

- help command
    - to view all commands

```
cargo help
```

- check command
    - check your prog. for err.
    - without producing any executable
    - faster than running the prog.

```
```

```
cargo check
```

```
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/one_three_hello_cargo$
cargo run
   Compiling one_three_hello_cargo v0.1.0 (/home/bali-
king/BaliGit/kintsugi-stack-rust/one_three_hello_cargo)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.10s
     Running `target/debug/one_three_hello_cargo`
Hello, world!
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/one_three_hello_cargo$
cargo build
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.00s
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/one_three_hello_cargo$
cargo check
    Checking one_three_hello_cargo v0.1.0 (/home/bali-
king/BaliGit/kintsugi-stack-rust/one_three_hello_cargo)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.02s
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/one_three_hello_cargo$
```

```
.
├── Cargo.lock
├── Cargo.toml
├── src
│   └── main.rs
└── target
    ├── CACHEDIR.TAG
    └── debug
        ├── build
        ├── deps
        │   ├── libone_three_hello_cargo-f9884884092cd48a.rmeta
        │   ├── one_three_hello_cargo-5885dd703046e3fc
        │   ├── one_three_hello_cargo-5885dd703046e3fc.d
        │   └── one_three_hello_cargo-f9884884092cd48a.d
        ├── examples
        ├── incremental
        │   ├── one_three_hello_cargo-10ah7hvrv4gzi
        │   │   ├── s-heho9i1rut-1ysdico-bwbrfy6ptxbomf5iwqz5vt3f0
        │   │   │   ├── dep-graph.bin
        │   │   │   ├── query-cache.bin
        │   │   │   └── work-products.bin
        │   │   └── s-heho9i1rut-1ysdico.lock
        │   └── one_three_hello_cargo-3dgwin0zbxstr
        │       ├── s-heho98ij63-039vdoh-143bk3qfw5zxnyx5otl9s0tja
        │       │   ├── 00ylhni9avwle6wyqpyzm6par.o
        │       │   ├── 19k6gm7hj98zo0jv2b5mu1std.o
        │       │   ├── 1jqdhkz0e02p777bbobcmna2j.o
        │       │   ├── 5fgvfmdk1vtvncsc4ze5a0wi9.o
        │       │   ├── 8z1o97dthkm4wl9qy6anckmmy.o
```

```
        │       │   ├── 9fwica1fdmiqw5oux5l4cedjc.o
        │       │   ├── dep-graph.bin
        │       │   ├── query-cache.bin
        │       │   └── work-products.bin
        │       └── s-heho98ij63-039vdoh.lock
        ├── one_three_hello_cargo
        └── one_three_hello_cargo.d
```

2. Programming a Guessing Game

```
Guess the Number !!!
Input Your Guess:
50
You Guessed: 50
TOO SMALL !!!
Input Your Guess:
25
You Guessed: 25
TOO SMALL !!!
Input Your Guess:
75
You Guessed: 75
TOO SMALL !!!
Input Your Guess:
90
You Guessed: 90
TOO SMALL !!!
Input Your Guess:
100
You Guessed: 100
TOO BIG !!!
Input Your Guess:
momo
Input Your Guess:
asdsadasdsadsdsdsdsdsdd
Input Your Guess:
95
You Guessed: 95
TOO BIG !!!
Input Your Guess:
93
You Guessed: 93
YOU WIN !!!
```

```rust
use std::{cmp::Ordering, io}; // io lib in scope

// Random Library
// to add deps "rand" package => add `deps = "version"` in `Cargo.toml` =>
cargo build
// [dependencies]
// rand = "0.5.5"
use rand::{Rand, Rng};

// colored library
// colored="2.0.0"
use colored::*;

// use `cargo run` or Run Button in Vsc at the main line( comes with
extension )
fn main() {
    // intro lines print
    println!("Guess the Number !!!"); // like python/c

    // Now Random Check is Left
```

```
    // // Random Library
    // // to add deps "rand" package => add `deps = "version"` in
`Cargo.toml` => cargo build
    // // [dependencies]
    // // rand = "0.5.5"
    // use rand::{Rand, Rng};
    let secret_nos = rand::thread_rng().gen_range(1,101); // lower limit is
inclusive, upper limit is exclusive
    // println!("Actual Number: {}", secret_nos);


    // to make game more interesting we can have game on loop to guess till
user guess the number correctly
    loop {

    println!("Input Your Guess:");

    // variable to store stuff
    // String, A type is Rust Standard library, utf-8, growable string
    // new() is associative func. static method, create empty string
    // Variables in Rust are DEFAULT IMMUTABLE, to make them mutable, use
mut keyword
    let mut guess = String::new(); // like java

    // io lib in scope
    // use std::io; // io lib in scope
    // .read_line method to read line
    // Result cases to 1. Ok() & 2. Err()
    io::stdin() // like java
        .read_line(&mut guess)
        .expect("Failed to Read Line"); // iff err comes, .expect() crash
program, and display message

    // Shadowing, we declare one variable (let mut guess = String::new();)
and then redeclare to convert the datatype but to preserve the value
    // .trim() remove whitespaces
    // .parse() helps to parse
    // let guess: u32 = guess.trim().parse().expect("Failed to Read
Line");// error handling strict by language // old way
    let guess: u32 = match guess.trim().parse(){
        Ok(num)=> num,
        Err(_)=> continue // `_` means catch all
        // to whatever any wrong input comes, continue the loop
    };// new way

    println!("You Guessed: {}",guess); // like c
    // Guess the Number !!!
    // Input Your Guess:
    // 12
    // You Guessed: 12

    // cmp::Ordering library
    // match guess.cmp(&secret_nos){
```

```
    //     Ordering::Equal => {print!("YOU WIN !!!");break;},// to
terminate after win is to break the loop // New way
    //     // Ordering::Equal => print!("YOU WIN !!!"), // Old way
    //     Ordering::Less => print!("TOO SMALL !!!"),
    //     Ordering::Greater => print!("TOO BIG !!!")
    // } // Old way no color

    // New Way, Color-ed :)
    match guess.cmp(&secret_nos){
    Ordering::Equal => {
        print!("{}","YOU WIN !!!".yellow());
        println!();// newline cosmetic code
        break;
    },
    Ordering::Less => print!("{}","TOO SMALL !!!".red()),
    Ordering::Greater => print!("{}","TOO BIG !!!".green())
    }

    println!();// newline cosmetic code

    // basic working
    // Guess the Number !!!
    // Input Your Guess:
    // 2
    // You Guessed: 2
    // Actual Number: 2
    // YOU WIN !!!

    }

}
```

- process
- initialize project

```
cargo new two_guessing_game
&& cd two_guessing_game
```

- `src/main.rs` code the basic logic

```
use std::io; // io lib in scope

fn main() {
    // intro lines print
    println!("Guess the Number !!!"); // like python/c
    println!("Input Your Guess:");
```

```
    // variable to store stuff
    // String, A type is Rust Standard library, utf-8, growable string
    // new() is associative func. static method, create empty string
    // Variables in Rust are DEFAULT IMMUTABLE, to make them mutable, use
mut keyword
    let mut guess = String::new(); // like java

    // io lib in scope
    // .read_line method to read line
    // Result cases to 1. Ok() & 2. Err()
    io::stdin() // like java
        .read_line(&mut guess)
        .expect("Failed to Read Line"); // iff err comes, .expect() crash
program, and display message


    println!("You Guessed {} !!!",guess); // like c

    // Guess the Number !!!
    // Input Your Guess:
    // 12
    // You Guessed 12
    //  !!!

    // Now Random Check is Left

}
```

- use `cargo run` or Run Button in Vsc at the main line(comes with extension)
- Now Random Check is Left
- to add deps "rand" package
    - add `deps = "version"` in `Cargo.toml`

```
[package]
name = "two_guessing_game"
version = "0.1.0"
edition = "2024"

[dependencies]
rand = "0.5.5"
```

- then

```
cargo build
```

```
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/two_guessing_game/src$
cargo build
```

```
   Compiling rand_core v0.4.2
   Compiling libc v0.2.178
   Compiling rand_core v0.3.1
   Compiling rand v0.5.6 # Gotcha
   Compiling two_guessing_game v0.1.0 (/home/bali-king/BaliGit/kintsugi-
stack-rust/two_guessing_game)
```

- then random number and check logic :

```
    // // Random Library
    // // to add deps "rand" package => add `deps = "version"` in
`Cargo.toml` => cargo build
    // // [dependencies]
    // // rand = "0.5.5"
    // use rand::{Rand, Rng};
    let secret_nos = rand::thread_rng().gen_range(1,101); // lower limit is
inclusive, upper limit is exclusive
    println!("Actual Number: {}", secret_nos);

    // cmp::Ordering library
    match guess.cmp(&secret_nos){
        Ordering::Equal => print!("YOU WIN !!!"),
        Ordering::Less => print!("TOO SMALL !!!"),
        Ordering::Greater => print!("TOO BIG !!!")
    }
```

- thus, the output is :

```
Guess the Number !!!
Input Your Guess:
2
You Guessed: 2
Actual Number: 2
YOU WIN !!!
```

- now guess logic is done
- to make game more interesting we can have game on loop to guess till user guess the number correctly
- put the guess input and match logic code in this `loop{ ... }`

```
// to make game more interesting we can have game on loop to guess till
user guess the number correctly
    loop {

    println!("Input Your Guess:");

    // variable to store stuff
```

```
    // String, A type is Rust Standard library, utf-8, growable string
    // new() is associative func. static method, create empty string
    // Variables in Rust are DEFAULT IMMUTABLE, to make them mutable, use
mut keyword
    let mut guess = String::new(); // like java

    // io lib in scope
    // use std::io; // io lib in scope
    // .read_line method to read line
    // Result cases to 1. Ok() & 2. Err()
    io::stdin() // like java
        .read_line(&mut guess)
        .expect("Failed to Read Line"); // iff err comes, .expect() crash
program, and display message

    // Shadowing, we declare one variable (let mut guess = String::new();)
and then redeclare to convert the datatype but to preserve the value
    // .trim() remove whitespaces
    // .parse() helps to parse
    // let guess: u32 = guess.trim().parse().expect("Failed to Read
Line");// error handling strict by language // old way
    let guess: u32 = match guess.trim().parse(){
        Ok(num)=> num,
        Err(_)=> continue // `_` means catch all
        // to whatever any wrong input comes, continue the loop
    };// new way

    println!("You Guessed: {}",guess); // like c
    // Guess the Number !!!
    // Input Your Guess:
    // 12
    // You Guessed: 12

    // cmp::Ordering library
    match guess.cmp(&secret_nos){
        Ordering::Equal => {print!("YOU WIN !!!");break;},// to terminate
after win is to break the loop // New way
        // Ordering::Equal => print!("YOU WIN !!!"), // Old way
        Ordering::Less => print!("TOO SMALL !!!"),
        Ordering::Greater => print!("TOO BIG !!!")
    }

    // basic working
    // Guess the Number !!!
    // Input Your Guess:
    // 2
    // You Guessed: 2
    // Actual Number: 2
    // YOU WIN !!!

    }
```

```
Guess the Number !!!
Input Your Guess:
20
You Guessed: 20
TOO SMALL !!!Input Your Guess:
30
You Guessed: 30
TOO SMALL !!!Input Your Guess:
40
You Guessed: 40
TOO SMALL !!!Input Your Guess:
60
You Guessed: 60
TOO SMALL !!!Input Your Guess:
80
You Guessed: 80
TOO SMALL !!!Input Your Guess:
90
You Guessed: 90
TOO BIG !!!Input Your Guess:
81
You Guessed: 81
TOO SMALL !!!Input Your Guess:
89
You Guessed: 89
TOO BIG !!!Input Your Guess:
85
You Guessed: 85
YOU WIN !!!Input Your Guess:
85
You Guessed: 85
YOU WIN !!!Input Your Guess:
YOU WIN !!!Input Your Guess:
exit

thread 'main' (49618) panicked at src/main.rs:46:6:
Failed to Read Line: ParseIntError { kind: InvalidDigit }
note: run with `RUST_BACKTRACE=1` environment variable to display a
backtrace
```

- now, to terminate after win is to break the loop:
  - convert `Ordering::Equal => print!("YOU WIN !!!")` to `Ordering::Equal => {print!("YOU WIN !!!");break;}`

```
    match guess.cmp(&secret_nos){
        Ordering::Equal => {print!("YOU WIN !!!");break;},// to terminate
after win is to break the loop // New way
        // Ordering::Equal => print!("YOU WIN !!!"), // Old way
        Ordering::Less => print!("TOO SMALL !!!"),
```

```
        Ordering::Greater => print!("TOO BIG !!!")
    }
```

- working

```
Guess the Number !!!
Input Your Guess:
50
You Guessed: 50
TOO BIG !!!Input Your Guess:
25
You Guessed: 25
TOO SMALL !!!Input Your Guess:
40
You Guessed: 40
TOO SMALL !!!Input Your Guess:
45
You Guessed: 45
YOU WIN !!!
```

- futher improvement: at wrong input ,the program `panick`'s
  - put match case handling at parsing of input string
  - convert `let guess: u32 = guess.trim().parse().expect("Failed to Read Line");// error handling strict by language` to this below

```
    // .trim() remove whitespaces
    // .parse() helps to parse
    // let guess: u32 = guess.trim().parse().expect("Failed to Read
Line");// error handling strict by language // old way
    let guess: u32 = match guess.trim().parse(){
        Ok(num)=> num,
        Err(_)=> continue // `_` means catch all
        // to whatever any wrong input comes, continue the loop
    };// new way
```

- Key Concept **Shadowing**: we declare one variable `let mut guess = String::new();` and then re-declare to convert the datatype but to preserve the value

```
let mut guess = String::new();

// Shadowing
let guess: u32 = match guess.trim().parse(){
        Ok(num)=> num,
        Err(_)=> continue
    };
```

- working, handling wrong inputs gracefully

```
Guess the Number !!!
Input Your Guess:
MOMO and Chutney
Input Your Guess:
50
You Guessed: 50
TOO SMALL !!!Input Your Guess:
75
You Guessed: 75
TOO BIG !!!Input Your Guess:
65
You Guessed: 65
TOO BIG !!!Input Your Guess:
55
You Guessed: 55
TOO SMALL !!!Input Your Guess:
53
You Guessed: 53
TOO SMALL !!!Input Your Guess:
51
You Guessed: 51
TOO SMALL !!!Input Your Guess:
52
You Guessed: 52
TOO SMALL !!!Input Your Guess:
54
You Guessed: 54
TOO SMALL !!!Input Your Guess:
58
You Guessed: 58
YOU WIN !!!
```

- further improvement: add colors
- add colored lib in `.toml` -> `colored="2.0.0"` & build it

```
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/two_guessing_game$
cargo build
   Compiling lazy_static v1.5.0
   Compiling colored v2.2.0
   Compiling two_guessing_game v0.1.0 (/home/bali-king/BaliGit/kintsugi-
stack-rust/two_guessing_game)
```

- color logic
    - too small -> Red
    - too big -> green
    - win -> yellow

- re-code match guess comparison check with text outputs `"YOU WIN !!!"` as values with color lib's methods `.red()`

```
// match guess.cmp(&secret_nos){
    //     Ordering::Equal => {print!("YOU WIN !!!");break;},// to
terminate after win is to break the loop // New way
    //     // Ordering::Equal => print!("YOU WIN !!!"), // Old way
    //     Ordering::Less => print!("TOO SMALL !!!"),
    //     Ordering::Greater => print!("TOO BIG !!!")
    // } // Old way no color

    // New Way, Color-ed :)
    match guess.cmp(&secret_nos){
    Ordering::Equal => {print!("{}","YOU WIN !!!".yellow());break;},
    Ordering::Less => print!("{}","TOO SMALL !!!".red()),
    Ordering::Greater => print!("{}","TOO BIG !!!".green())
    }

    println!();// newline cosmetic code
```

- some cosmetic changes

```
 match guess.cmp(&secret_nos){
    Ordering::Equal => {
        print!("{}","YOU WIN !!!".yellow());
        println!();// newline cosmetic code
        break;
    },
    Ordering::Less => print!("{}","TOO SMALL !!!".red()),
    Ordering::Greater => print!("{}","TOO BIG !!!".green())
    }

    println!();// newline cosmetic code
```

- working

```
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/two_guessing_game$ cargo run
warning: `two_guessing_game` (bin "two_guessing_game") generated 2 warnings (run `cargo fix --bin "two_guessing_game" -p two_guessing_game` to a
pply 1 suggestion)
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.10s
     Running `target/debug/two_guessing_game`
Guess the Number !!!
Input Your Guess:
50
You Guessed: 50
TOO BIG !!!
Input Your Guess:
100
You Guessed: 100
TOO BIG !!!
Input Your Guess:
25
You Guessed: 25
TOO BIG !!!
Input Your Guess:
10
You Guessed: 10
TOO SMALL !!!
Input Your Guess:
15
You Guessed: 15
TOO SMALL !!!
Input Your Guess:
20
You Guessed: 20
TOO SMALL !!!
Input Your Guess:
25
You Guessed: 25
TOO BIG !!!
Input Your Guess:
22
You Guessed: 22
YOU WIN !!!
bali-king@war-machine:~/BaliGit/kintsugi-stack-rust/two_guessing_game$
```

3. Common Programming Concepts

3.1. Variables and Mutability

```rust
// use std::io;
fn main(){
    // Variables and Mutability

    // Vars in Rust are immutable by default
    // also will throw warning as it's never used in code
    // warning: unused variable: `val1`
    let val1 = 10;

    // you cannot re-assign immutable entity
    // error[E0384]: cannot assign twice to immutable variable `val1`
    // val1 = 20; // No

    // Mutable Variable, just by declare `mut` during initialization
    // also will throw warning as it's(this assignment) never used in code(
even thou we used it to print, but print uses the 2nd assigned value, not
1st )
    // warning: value assigned to `val2` is never read
    let mut val2 = 20;

    // Re-assigned Successfully
    // also not throw warning as we use this assignment (i.e. 2nd) to
println
    val2 = 30;
    println!("{}",val2);
    // 30
```

```
    // unused variable warning fix
    // if this is intentional, prefix it with an underscore: `_val1`
    let _val3 = 10;

    // Constants
    // Contant values

    // - Constant naming convention:
    //  - ALL CAPS
    //  - underscores(_) replacing spaces( )
    // eg: "Fixed value"(normal word intention) -> FIXED_VALUE" (Constant
Name)
    const FIXED_VALUE: u32 = 1;// YES

    // can't use `mut` keyword in constants
    // error: const globals cannot be mutable
    // const mut FIXED_VALUE =1; // No

    // must be datatype annotated
    // error: missing type for `const` item, help: provide a type for the
constant
    // const FIXED_VALUE =1; // No

    // numeric literals readability feature
    // while storing numbers, we can use underscores as commas for
readability
    // eg: 1000 = 1___000 = 10_00__  at rust
    // eg: _1000 is wrong at rust
    // purpoes example; eg: 1lakh, 100000, we use commas in english for
readability 1,00,000.
    // at rust; to improve readability; 100000 = 1_00_000
    const FIXED_VALUE_1: u128 = 1_00_000 ;
    println!("{}",FIXED_VALUE_1); // 100000

    // constants may be set only to a constant expression
    // , not the result of a value that could only be computed at runtime.
    const FIXED_VALUE_2: u128 = 1+ FIXED_VALUE_1; // allowed, computation
done at compile time
    println!("{}", FIXED_VALUE_2);
    const FIXED_VALUE_3: u128 = 100*20000 + 10; // allowed, computation
done at compile time
    println!("{}", FIXED_VALUE_3);

    // error[E0435]: attempt to use a non-constant value in a constant
    // computation done at runtime not allowed at const. assign

    // // example for computation at runtime
    // let mut input:u128 ;
    // io::stdin()
    //     .read_line(&mut input)
    //     .expect("Failed to read input");
    // // const FIXED_VALUE_4: u128 = 1+ input; // NO, not allowed
```

```
    // Shadowing
    // Way to create a new variable using existing name
    // helps to preserve mutability and even update the value
    let x =1;// x is immutable,
    // x=2; // NO, error[E0384]: cannot assign twice to immutable variable
`x`
    let x=2;// YES, Shadowing, x is still immutable, mutability preserved
    println!("{}",x);// 2
    let x="two";// YES, Shadowing, x is still immutable, mutability
preserved, and typecasting
    println!("{}",x);// two
    let x=3;// YES, Shadowing, x is still immutable, mutability preserved
    println!("{}",x);// 3

    // purpoes example: to typecast data
    let x = x.to_string(); // Shadowing to Typecast int -> string
    println!("{}",x);// 3

    // clone()
    // we can't use x as x is chaged as string ops.
    // so we use copy of x to play with it
    // println!("{}",x+"10");// 310 // NO, it will work at this line, after
this line if used somewhere it will throw error, error[E0382]: borrow of
moved value: `x`
    println!("{}",x.clone()+"10");// 310 // YES

}
```

- Variables and Mutability

```
    // Variables and Mutability

    // Vars in Rust are immutable by default
    // also will throw warning as it's never used in code
    // warning: unused variable: `val1`
    let val1 = 10;

    // you cannot re-assign immutable entity
    // error[E0384]: cannot assign twice to immutable variable `val1`
    // val1 = 20; // No

    // Mutable Variable, just by declare `mut` during initialization
    // also will throw warning as it's(this assignment) never used in code(
even thou we used it to print, but print uses the 2nd assigned value, not
1st )
    // warning: value assigned to `val2` is never read
    let mut val2 = 20;

    // Re-assigned Successfully
    // also not throw warning as we use this assignment (i.e. 2nd) to
println
```

```
    val2 = 30;
    println!("{}",val2);
    // 30

    // unused variable warning fix
    // if this is intentional, prefix it with an underscore: `_val1`
    let _val3 = 10;
```

- Constants

```
    // Constants
    // Contant values

    // - Constant naming convention:
    //  - ALL CAPS
    //  - underscores(_) replacing spaces( )
    // eg: "Fixed value"(normal word intention) -> FIXED_VALUE" (Constant
Name)
    const FIXED_VALUE: u32 = 1;// YES

    // can't use `mut` keyword in constants
    // error: const globals cannot be mutable
    // const mut FIXED_VALUE =1; // No

    // must be datatype annotated
    // error: missing type for `const` item, help: provide a type for the
constant
    // const FIXED_VALUE =1; // No

    // numeric literals readability feature
    // while storing numbers, we can use underscores as commas for
readability
    // eg: 1000 = 1___000 = 10_00__  at rust
    // eg: _1000 is wrong at rust
    // purpoes example; eg: 1lakh, 100000, we use commas in english for
readability 1,00,000.
    // at rust; to improve readability; 100000 = 1_00_000
    const FIXED_VALUE_1: u128 = 1_00_000 ;
    println!("{}",FIXED_VALUE_1); // 100000

    // constants may be set only to a constant expression
    // , not the result of a value that could only be computed at runtime.
    const FIXED_VALUE_2: u128 = 1+ FIXED_VALUE_1; // allowed, computation
done at compile time
    println!("{}", FIXED_VALUE_2);
    const FIXED_VALUE_3: u128 = 100*20000 + 10; // allowed, computation
done at compile time
    println!("{}", FIXED_VALUE_3);

    // error[E0435]: attempt to use a non-constant value in a constant
    // computation done at runtime not allowed at const. assign
```

```
    // // example for computation at runtime
    // let mut input:u128 ;
    // io::stdin()
    //     .read_line(&mut input)
    //     .expect("Failed to read input");
    // // const FIXED_VALUE_4: u128 = 1+ input; // NO, not allowed
```

- Shadowing

```
    // Shadowing
    // Way to create a new variable using existing name
    // helps to preserve mutability and even update the value
    let x =1;// x is immutable,
    // x=2; // NO, error[E0384]: cannot assign twice to immutable variable
`x`
    let x=2;// YES, Shadowing, x is still immutable, mutability preserved
    println!("{}",x);// 2
    let x="two";// YES, Shadowing, x is still immutable, mutability
preserved, and typecasting
    println!("{}",x);// two
    let x=3;// YES, Shadowing, x is still immutable, mutability preserved
    println!("{}",x);// 3

    // purpoes example: to typecast data
    let x = x.to_string(); // Shadowing to Typecast int -> string
    println!("{}",x);// 3

    // clone()
    // we can't use x as x is chaged as string ops.
    // so we use copy of x to play with it
    // println!("{}",x+"10");// 310 // NO, it will work at this line, after
this line if used somewhere it will throw error, error[E0382]: borrow of
moved value: `x`
    println!("{}",x.clone()+"10");// 310 // YES
```

- Keywords
  - set of keywords that are reserved for use by the language only, much as in other languages.
  - Keep in mind that you cannot use these words as names of variables or functions.
  - https://doc.rust-lang.org/stable/book/appendix-01-keywords.html

3.2. Data Types

```
fn main(){
    // Data Types
    // Scaler datatypes represent single value
    // Compound datatypes represent group of values
```

```
    // Scaler datatypes
    // Integers
    // Floating-point numbers
    // Booleans
    // Character

    // Integers
    // numbers without fractional component
    // Signed:      either +ve/-ve integers
    // unsigned:    only +ve integers

    // Integer Types in Rust
    // Length   Signed  Unsigned
    // 8-bit    i8  u8
    // 16-bit   i16 u16
    // 32-bit   i32 u32
    // 64-bit   i64 u64
    // 128-bit  i128    u128
    // Architecture-dependent   isize   usize

    // Architecture-dependent: the isize and usize types depend on the
architecture of the computer your program is running on: 64 bits if you're
on a 64-bit architecture and 32 bits if you're on a 32-bit architecture.

    // i32 is default in rust integer datatype

    let a = 98_222; // Decimal
    let b = 0xff; // Hex
    let c = 0o77; // Octal
    let d = 0b1111_0000; // Binary
    let e = b'A'; // Byte ( u8 only )

    // Integer Overflow
    let f: u8 = 255; // max value of u8
    // let g: u8 = 256; // Integer Overflow
    // at debug build -> Rust Panics (error: literal out of range for `u8`)
    // at release build(compiling in release mode with the --release flag)
-> Rust performs 2's Complement Wrapping
    // Rust performs 2's Complement Wrapping => number exceeding max value
will wrap around minimum value
    // 256 -> 0+1 -> 1
    // let h: u8 = 260; // 260 -> 0+5 -> 5

    // btw
    // cargo build --release # release flag
    // rustc -O main.rs # optimization flag

    // Floating-point numbers
    // Decimal values
    let i = 6.8; // default is f64, 64 bit ; double precision floating
point number
    let j:f32 = 6.9; // f32
    // The default type is f64 because on modern CPUs, it's roughly the
same speed as f32 but is capable of more precision.
```

```
    // All floating-point types are signed.
    // Floating-point numbers are represented according to the IEEE-754
standard.

    // Numeric Operations
    let sum = 5+5333; // Addition
    let div_quotient = 55/2; // Division Quotient
    let div_truncated = -5/3; // Division Truncated
    // Results in -1
    let remainder = 10%2; // Remainder
    let mul = 55.5 * 100.222; // Multiplication
    let sub = 0.0 - 10.99; // Subtraction

    // Boolean
    // two possible values: true and false.
    // Booleans are one byte in size.
    let k = true;
    let l:bool = false;

    // Character
    // Rust’s char type is 4 bytes in size
    // Unicode Values
    let c = 'z';
    let z: char = 'ℤ';
    let heart_eyed_cat = '😻';

    // Compund Types
    // Represent a group of Values
    // The Tuple Type
    // The Array Type

    // The Tuple Type ()
    // Fixed Size Array
    // Different Datatype Elements
    let tup = ("Hello World",102_00, 1001.11,'😻', true);
    let tup1: (i32, f64, u8) = (500, 6.4, 1);
    // Tuple Access Ways
    // Tuple Access Ways: destructuring & dot notation
    // destructuring way
    let (val1,val2,val3) = tup1;
    // dot notation way
    let val1_1 = tup1.0;
    // tuple index start from 0
    let val2_1 = tup1.1;
    let val3_1 = tup1.2;

    // The Array Type []
    // Fixed Size Array
    // Same Datatype Elements
    let a = [1,2,3];
    let b = [1.01,2.12212,3.0];
    // let array_out_of_bound_err_exception_eg = b[3]; // NO, index out of
bounds
    let months = ["January", "February", "March", "April", "May", "June",
```

```
"July",
                "August", "September", "October", "November", "December"];
    let curr_month = months[0]; // Array Access
    let a = [-20; 3]; // Array Initialize Multiple at once
[Value;Occurences]
    // [-20, -20, -20]

    // Rust Prevents us to do anything Memory Unsafe !!!


}
```

- types of data types

```
    // Data Types
    // Scaler datatypes represent single value
    // Compound datatypes represent group of values
```

- types of Scaler data types

```
    // Scaler datatypes
    // Integers
    // Floating-point numbers
    // Booleans
    // Character
```

- Integers

```
    // Integers
    // numbers without fractional component
    // Signed:      either +ve/-ve integers
    // unsigned:    only +ve integers

    // Integer Types in Rust
    // Length   Signed  Unsigned
    // 8-bit    i8  u8
    // 16-bit   i16 u16
    // 32-bit   i32 u32
    // 64-bit   i64 u64
    // 128-bit  i128    u128
    // Architecture-dependent   isize   usize

    // Architecture-dependent: the isize and usize types depend on the
architecture of the computer your program is running on: 64 bits if you're
on a 64-bit architecture and 32 bits if you're on a 32-bit architecture.
```

```
    // i32 is default in rust integer datatype

    let a = 98_222; // Decimal
    let b = 0xff; // Hex
    let c = 0o77; // Octal
    let d = 0b1111_0000; // Binary
    let e = b'A'; // Byte ( u8 only )

    // Integer Overflow
    let f: u8 = 255; // max value of u8
    let g: u8 = 256; // Integer Overflow
    // at debug build -> Rust Panics (error: literal out of range for `u8`)
    // at release build(compiling in release mode with the --release flag)
-> Rust performs 2's Complement Wrapping
    // Rust performs 2's Complement Wrapping => number exceeding max value
will wrap around minimum value
    // 256 -> 0+1 -> 1
    let h: u8 = 260; // 260 -> 0+5 -> 5
```

- release & optimization flag

```
    // cargo build --release # release flag
    // rustc -O main.rs # optimization flag
```

- Floating-point numbers

```
    // Floating-point numbers
    // Decimal values
    let i = 6.8; // default is f64, 64 bit ; double precision floating
point number
    let j:f32 = 6.9; // f32
    // The default type is f64 because on modern CPUs, it’s roughly the
same speed as f32 but is capable of more precision.
    // All floating-point types are signed.
    // Floating-point numbers are represented according to the IEEE-754
standard.
```

- Numeric Operations

```
    // Numeric Operations
    let sum = 5+5333; // Addition
    let div_quotient = 55/2; // Division Quotient
    let div_truncated = -5/3; // Division Truncated
    // Results in -1
    let remainder = 10%2; // Remainder
    let mul = 55.5 * 100.222; // Multiplication
    let sub = 0.0 - 10.99; // Subtraction
```

- Boolean

```
// Boolean
// two possible values: true and false.
// Booleans are one byte in size.
let k = true;
let l:bool = false;
```

- Character

```
// Character
// Rust’s char type is 4 bytes in size
// Unicode Values
let c = 'z';
let z: char = 'ℤ';
let heart_eyed_cat = '😻';
```

- Compund Types

```
// Compund Types
// Represent a group of Values
// The Tuple Type
// The Array Type
```

- Tuple

```
// The Tuple Type ()
// Fixed Size Array
// Different Datatype Elements
let tup = ("Hello World",102_00, 1001.11,'😻', true);
let tup1: (i32, f64, u8) = (500, 6.4, 1);
// Tuple Access Ways
// Tuple Access Ways: destructuring & dot notation
// destructuring way
let (val1,val2,val3) = tup1;
// dot notation way
let val1_1 = tup1.0;
// tuple index start from 0
let val2_1 = tup1.1;
let val3_1 = tup1.2;
```

- Array

```
    // The Array Type []
    // Fixed Size Array
    // Same Datatype Elements
    let a = [1,2,3];
    let b = [1.01,2.12212,3.0];
    // let array_out_of_bound_err_exception_eg = b[3]; // NO, index out of
bounds
    let months = ["January", "February", "March", "April", "May", "June",
"July",
              "August", "September", "October", "November", "December"];
    let curr_month = months[0]; // Array Access
    let a = [-20; 3]; // Array Initialize Multiple at once
[Value;Occurences]
    // [-20, -20, -20]

    // Rust Prevents us to do anything Memory Unsafe !!!
```

3.3. Functions

```
fn main(){
    func_1(); // simple function call
    // Hello World !!!
    func_2(22,"Kintsugi-programmer"); // function call with arguements
    // Hi Kintsugi-programmer, you type 22
    println!("{}",func_3(1,10));
    println!("{}",func_3_1(1,10));
    let sum = func_3_2(1,10);
    println!("{}",sum);
    let val = five();
    println!("{}",val);
}

// Functions
// Functions Convention in Rust : snake_case
// in rust, Function code is either
// statement(do action and no return)
// or expression(do action and return some value)
fn func_1(){println!("Hello World !!!");} // simple function // Statement
fn func_2(x:u128, y:&str){println!("Hi {}, you type {}",y,x);} // Statement

// return, in Expression Functions
fn five() -> i32 {
    5
}

// return ways

// return way 1
fn func_3(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi"); // statement
```

```
    let sum = x+y;
    return sum; // return keyword
}

// return way 2
fn func_3_1(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi");
    let sum = x+y;
    sum // remove return keyword AND remove semicolon to return this value
    // rust func will return this last expression
    // this not statement, as we remove ;, and make it expression
}

fn func_3_2(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi");
    x+y // more simplified version
}

// Hello World !!!
// Hi Kintsugi-programmer, you type 22
// Hi
// 11
// Hi
// 11
// Hi
// 11
// 5
```

- Functions

```
// Functions
// Functions Convention in Rust : snake_case
fn func_1(){println!("Hello World !!!");} // simple function
fn func_2(x:u128, y:&str){println!("Hi {}, you type {}",y,x);}
```

- Statement and Expressions

```
// in rust, Function code is either
// statement(do action and no return)
// or expression(do action and return some value)
fn func_1(){println!("Hello World !!!");} // simple function // Statement
    // func_1(); // simple function call
    // Hello World !!!
fn func_2(x:u128, y:&str){println!("Hi {}, you type {}",y,x);} // Statement
    func_2(22,"Kintsugi-programmer");
    // function call with arguements
    // Hi Kintsugi-programmer, you type 22
// return, in Expression Functions
```

```
fn five() -> i32 {
    5
}
    // let val = five();
    // println!("{}",val);
    // 5
```

- Return Ways

```
// return ways

// return way 1
fn func_3(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi"); // statement
    let sum = x+y;
    return sum; // return keyword
}

// return way 2
fn func_3_1(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi");
    let sum = x+y;
    sum // remove return keyword AND remove semicolon to return this value
    // rust func will return this last expression
    // this not statement, as we remove ;, and make it expression
}

fn func_3_2(x:u128, y:u128) -> u128{ // specify return type
    println!("Hi");
    x+y // more simplified version
}
```

3.4. Comments

```
// comments
// programmers leave comments in their source code that the compiler will
ignore but that people reading the source code may find useful.

fn main(){
    // Comment Ways
    // way 1 : use double slashes (//)
    /*
    way 2:
    Block Comments
    For Multilines
    */
    // other ways : doc comments, refer to 14.2.Publishing a Crate to
Crates.io
```

```
    // Multiline Convention
    // So we're doing something complicated here, long enough that we need
    // multiple lines of comments to do it! Whew! Hopefully, this comment
will
    // explain what's going on.

    // compiler ignores comments, no-matter what they are
    // println!("Rust is Worst Language");
    // above code is wrong and we don't want to use as code from compiler
but we don't wanna delete too as we don't know future !!!
    println!("Rust is God Language"); // Correct One :)
    // Rust is God Language
    /*
        println!("Rust is Worst Language");


    */
    // Also at Rust's Features, You can Make Documentations Easily within
Code, refer to 14.2.Publishing a Crate to Crates.io

}
```

- Comments

```
// comments
// programmers leave comments in their source code that the compiler will
ignore but that people reading the source code may find useful.
```

- Multiline Convention

```
    // Multiline Convention
    // So we're doing something complicated here, long enough that we need
    // multiple lines of comments to do it! Whew! Hopefully, this comment
will
    // explain what's going on.
```

- Comment Ways

```
    // Comment Ways
    // way 1 : use double slashes (//)
    /*
    way 2:
    Block Comments
    For Multilines
    */
```

```
    // other ways : doc comments, refer to 14.2.Publishing a Crate to
Crates.io
```

- Comments Essence

```
    // compiler ignores comments, no-matter what they are
    // println!("Rust is Worst Language");
    // above code is wrong and we don't want to use as code from compiler
but we don't wanna delete too as we don't know future !!!
    println!("Rust is God Language"); // Correct One :)
    // Rust is God Language
    /*
        println!("Rust is Worst Language");


    */
    // Also at Rust's Features, You can Make Documentations Easily within
Code, refer to 14.2.Publishing a Crate to Crates.io
```

3.5. Control Flow

```

```

-

End-of-File

The KintsugiStack repository, authored by Kintsugi-Programmer, is less a comprehensive resource and more an Artifact of Continuous Research and Deep Inquiry into Computer Science and Software Engineering. It serves as a transparent ledger of the author's relentless pursuit of mastery, from the foundational algorithms to modern full-stack implementation.

> Made with 💚 Kintsugi-Programmer