Table

Description automatically generated

let decimal:f64 = 65.4321; let integer = decimal as u8

let emoji = "\u{1f600}"; // => 😀

let array: [i64; 6] = [92,97,98,99,98,94];

let array: [[i64; 6] ;2] = [

[1,2,3,4,5,6],

[6,5,4,3,2,1]];

let some\_vector = vec1,2,3,4,5];

let tokens: Vec<&str> = color.split(",").collect();

let mut num: Vec<i32> = Vec::new();

num.push(1);

d.pop();

let v = vec![1, 2, 3, 4, 5];

let v2 = v.iter().map(|x| x \* x).collect::<Vec<i32>>() // iter breaks vec or array or string to pieces then map will apply function to each element. Lastly collect will put them back to vec that contains i32

let empty\_string = String::new();

"Hello".to\_string();

String::from("SE Chan");

university.len(); // string and vec

"Hello World".to\_string().replace("Hello", "Hi");

string\_1.trim(); // Removes leading and trailing spaces

string.split\_whitespace() // Splits string into iterator

string.split(",") // same but can choose

string.chars()

rand.capacity() // Calculates the capacity of the string in bytes. And vec

name.contains("Element") // returns bool

string.push\_str("How are you doing??"); // push the text into the var

&string[0..5]; // slices (same with vec) -> &v[0..5] (0..5 means 0 1 2 3 4)

format!("{a:.3} {b:?}")

#[derive(Debug)]

struct info {

name:String,

age:i32,

}

impl Point { // imagine there was struct point

fn new(x: f64, y: f64) -> Point {

Point { x: x, y: y }

}

}

let p = Point::new(1.0, 2.0);

let inf = info{

name:chanasorn.to\_string(),

age:12,

};

#[derive(debug)]

enum individual{ // acts like data type (string, i32)

human(info, String), // can put any shit here

orc(info, String),

elf(info,String)

}

let men = individual::human(inf, “this dude retard”.to\_string());

let women = individual::elf(inf, “big boba”.to\_string()); // cant inf here cuz inf’s value was moved to men already. Just showing how enum works.

//Wont write impl for enum cuz too long. Go read urself

enum Option<T> {

None,

Some(T),

}

&& ||

if x == 5 { 10 } else { 15 };

for mut i in 0..3 {}

loop {}

while x > 3 {}

match day\_of\_week {

1 => {

Println!(“sgdg”);

}

2 => {

Println!(“sgdg”);

}

\_ => {

Println!(“sgdg”);

}

}

fn dsfs() -> {}

fn print\_arrays(mut array: [i32; 5]) -> [i32; 5] {} // array as arg and returning arrays

fn returnTuple() -> (i32, String){ return (32, “dgdgf”.tostring();}

let (theanswer, fireship) = returnTuple(); // returning tuple so u get multiple variables

use std::io;

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read line"); // convert input to integer

let input: u32 = input.trim().parse().expect("Please type a number!");

use std::fs::File;

use std::io::Read;

let mut input = String::new();

File::open("input.txt").unwrap().read\_to\_string(&mut input).unwrap();

use std::fs::File;

use std::io::Write;

let mut file = File::create("output.txt").unwrap();

write!(file, "Hello, world!").unwrap();

:b = binary

:o = octal

:x = hex

println!("{number:>5}", number = 1); // 5 white spaces and a "1" }

println!("{number:0>5}", number = 1);

![Graphical user interface, text, application, email

Description automatically generatedText

Description automatically generated