Contents

List of Figures

Chapter 1

New Chapter

```
use std::clone;

struct GameInfo {
hours_played:  u16,
skill_level:  String,
played_since:  String,
}

enum GamesData {
Terraria(GameInfo),
Undertale(GameInfo),
DDLC(GameInfo),
}

struct User {
favoriteGame:  GamesData,
username:  String,
premium:  bool,
}

fn main() {
let terraria = GamesData::Terraria(GameInfo {
hours_played:  12,
skill_level:  String::from( "Mediocre" ),
played_since:  String::from( "2018" ),
});

let user1 = User {
favoriteGame:  terraria,
username:  "vbvnyk" .to_string(),
premium:  false,
};
```

```
match user1.favoriteGame {
GamesData::Terraria(gameInfo) => {
println!( " {  } likes to play {  }.  His skill level at the game is {  } and has been
playing since {  }" , user1.username, "terraria" .to_string(), gameInfo.skill_level,
gameInfo.hours_played );
}
GamesData::DDLC(gameInfo) => {
println!( " {  } likes to play {  }.  His skill level at the game is {  } and has been
playing since {  }" , user1.username, "terraria" .to_string(), gameInfo.skill_level,
gameInfo.hours_played );
}
GamesData::Undertale(gameInfo) => {
println!( " {  } likes to play {  }.  His skill level at the game is {  } and has been
playing since {  }" , user1.username, "terraria" .to_string(), gameInfo.skill_level,
gameInfo.hours_played );
}
}

let opt:  Option<String> = Some(String::from( "Hello world" ));

match &opt {
// _ became s
Some(s) => println!( "Some:  {  }" , s),
None => println!( "None!" ),
};
println!( " { :?  }" , opt);

let submitted:  Option<i8> = Some(30);
if let Some(score) = submitted {
println!( "Your assignment score is { score }" );
} else {
println!( "You still have to submit your assignment" );
}
}

 [ package ]
name = "enums"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[ dependencies ]
```

```
use rand::  { self, Rng };
use std::io;

fn main() {
let mut rng = rand::thread_rng();
let secret_number = rng.gen_range(1..10);
// print!( "The random floating number generated is {  }" , secret_number);

loop {
let mut guess = String::new();
println!( "Enter your guess:  " );
io::stdin()
.read_line(&mut guess)
.expect( "Please enter a valid number" );

let guess:  i32 = match guess.trim().parse() {
Ok(num) => num,
Err(_) => {
print!( "There was an error parsing your number, try again." );
continue;
}
};

match guess.cmp(&secret_number) {
std::cmp::Ordering::Equal => {
println!( "You won!!" );
break;
}
std::cmp::Ordering::Greater => println!( "Your guess was higher" ),
std::cmp::Ordering::Less => println!( "Your guess was lower" ),
}
}

println!( "Congratulations, you've won the game" );
}
```