패턴과 매칭

2020.4

패턴이 사용되는 곳

```
match 값 {
    패턴 => 표현,
    패턴 => 표현,
    패턴 => 표현,
}
```

모든 경우의 수를 다 표현해야 함.
_ 표현이 유용함.

```
fn main() {
    let favorite_color: Option<&str> = None;
    let is_tuesday = false;
    let age: Result<u8, _> = "34".parse();

    if let Some(color) = favorite_color {
        println!("Using your favorite color, {}, as the background", color);
    } else if is_tuesday {
        println!("Tuesday is green day!");
    } else if let Ok(age) = age {
        if age > 30 {
            println!("Using purple as the background color");
        } else {
            println!("Using orange as the background color");
        }
    } else {
        println!("Using blue as the background color");
    }
}
```

```
let mut stack = Vec::new();

stack.push(1);
stack.push(2);
stack.push(3);

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

```
let v = vec!['a', 'b', 'c'];

for (index, value) in v.iter().enumerate() {
    println!("{} is at index {}", value, index);
}
```

```
let PATTERN = EXPRESSION;
```

```
let (x, y, z) = (1, 2, 3);
```

```
fn print_coordinates(&(x, y): &(i32, i32)) {
    println!("Current location: ({}, {})", x, y);
}

fn main() {
    let point = (3, 5);
    print_coordinates(&point);
}
```

반증 가능성

반증불가 패턴 : 어떠한 값에도 대응되는 패턴
예) let x = 5;

반증 가능 패턴 : 주어진 값에 대응이 실패할 수 있다.

예) let Some(x) = a_value;

```
error[E0005]: refutable pattern in local binding: `None` not covered
 -->
  |
3 | let Some(x) = some_option_value;
  |     ^^^^^^^ pattern `None` not covered
```

```
if let Some(x) = some_option_value {
    println!("{}", x);
}
```

```
if let x = 5 {
    println!("{}", x);
};
```

```
error[E0162]: irrefutable if-let pattern
 --> <anon>:2:8
  |
2 | if let x = 5 {
  |        ^ irrefutable pattern
```

패턴 문법

```
let x = 1;

match x {
    1 => println!("one"),
    2 => println!("two"),
    3 => println!("three"),
    _ => println!("anything"),
}
```
리터럴 매칭

```
fn main() {
    let x = Some(5);
    let y = 10;

    match x {
        Some(50) => println!("Got 50"),
        Some(y) => println!("Matched, y = {:?}", y),
        _ => println!("Default case, x = {:?}", x),
    }

    println!("at the end: x = {:?}, y = {:?}", x, y);
}
```
명명 변수 매칭

```
let x = 1;

match x {
    1 | 2 => println!("one or two"),
    3 => println!("three"),
    _ => println!("anything"),
}
```
다중 패턴

```
let x = 5;

match x {
    1 ... 5 => println!("one through five"),
    _ => println!("something else"),
}
```
1,2,3,4,5

```
let x = 'c';

match x {
    'a' ... 'j' => println!("early ASCII letter"),
    'k' ... 'z' => println!("late ASCII letter"),
    _ => println!("something else"),
}
```

```
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let p = Point { x: 0, y: 7 };

    let Point { x: a, y: b } = p;
    assert_eq!(0, a);
    assert_eq!(7, b);
}
```

```
struct Point {
    x: i32,
    y: i32,
}

fn main() {
    let p = Point { x: 0, y: 7 };

    let Point { x, y } = p; // x: x, y: y
    assert_eq!(0, x);
    assert_eq!(7, y);
}
```

```
fn main() {
    let p = Point { x: 0, y: 7 };

    match p {
        Point { x, y: 0 } => println!("On the x axis at {}", x),
        Point { x: 0, y } => println!("On the y axis at {}", y),
        Point { x, y } => println!("On neither axis: ({}, {})", x, y),
    }
}
```

```
let points = vec![
    Point { x: 0, y: 0 },
    Point { x: 1, y: 5 },
    Point { x: 10, y: -3 },
];

let sum_of_squares: i32 = points
    .iter()
    .map(|&Point { x, y }| x * x + y * y)
    .sum();
```

패턴 문법

```
enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
    ChangeColor(i32, i32, i32),
}

fn main() {
    let msg = Message::ChangeColor(0, 160, 255);

    match msg {
        Message::Quit => {
            println!("The Quit variant has no data to destructure.")
        },
        Message::Move { x, y } => {
            println!(
                "Move in the x direction {} and in the y direction {}",
                x,
                y
            );
        }
        Message::Write(text) => println!("Text message: {}", text),
        Message::ChangeColor(r, g, b) => {
            println!(
                "Change the color to red {}, green {}, and blue {}",
                r,
                g,
                b
            )
        }
    }
}
```

```
fn foo(_: i32, y: i32) {
    println!("This code only uses the y parameter: {}", y);
}

fn main() {
    foo(3, 4);
}
```

```
let mut setting_value = Some(5);
let new_setting_value = Some(10);

match (setting_value, new_setting_value) {
    (Some(_), Some(_)) => {
        println!("Can't overwrite an existing customized value");
    }
    _ => {
        setting_value = new_setting_value;
    }
}

println!("setting is {:?}", setting_value);
```

```
let ((feet, inches), Point {x, y}) = ((3, 10), Point { x: 3, y: -10 });
```

패턴 문법

```rust
struct Point {
    x: i32,
    y: i32,
    z: i32,
}

let origin = Point { x: 0, y: 0, z: 0 };

match origin {
    Point { x, .. } => println!("x is {}", x),
}
```

```rust
fn main() {
    let numbers = (2, 4, 8, 16, 32);

    match numbers {
        (first, .., last) => {
            println!("Some numbers: {}, {}", first, last);
        },
    }
}
```

```rust
fn main() {
    let numbers = (2, 4, 8, 16, 32);

    match numbers {
        (.., second, ..) => {
            println!("Some numbers: {}", second)
        },
    }
}
```

```
error: `..` can only be used once per tuple or tuple struct pattern
 --> src/main.rs:5:22
  |
5 |         (.., second, ..) => {
  |                      ^^
```

```rust
let robot_name = Some(String::from("Bors"));

match robot_name {
    Some(name) => println!("Found a name: {}", name),
    None => (),
}

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

```rust
let robot_name = Some(String::from("Bors"));

match robot_name {
    Some(ref name) => println!("Found a name: {}", name),
    None => (),
}

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

```rust
let mut robot_name = Some(String::from("Bors"));

match robot_name {
    Some(ref mut name) => *name = String::from("Another name"),
    None => (),
}

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

패턴 문법

```
let num = Some(4);

match num {
    Some(x) if x < 5 => println!("less than five: {}", x),
    Some(x) => println!("{}", x),
    None => (),
}
```

```
fn main() {
    let x = Some(5);
    let y = 10;

    match x {
        Some(50) => println!("Got 50"),
        Some(n) if n == y => println!("Matched, n = {:?}", n),
        _ => println!("Default case, x = {:?}", x),
    }

    println!("at the end: x = {:?}, y = {:?}", x, y);
}
```

```
let x = 4;
let y = false;

match x {
    4 | 5 | 6 if y => println!("yes"),
    _ => println!("no"),
} // 4또는 5또는 6이고 true
```

```
enum Message {
    Hello { id: i32 },
}

let msg = Message::Hello { id: 5 };

match msg {
    Message::Hello { id: id_variable @ 3...7 } => {
        println!("Found an id in range: {}", id_variable)
    },
    Message::Hello { id: 10...12 } => {
        println!("Found an id in another range")
    },
    Message::Hello { id } => {
        println!("Found some other id: {}", id)
    },
} // 3부터 7 사이의 값을 변수 id_variable에 넣어준다.
```

Q & A