**Practical Programming** 

# Rust : Pattern Matching



#### **David Bouchet**

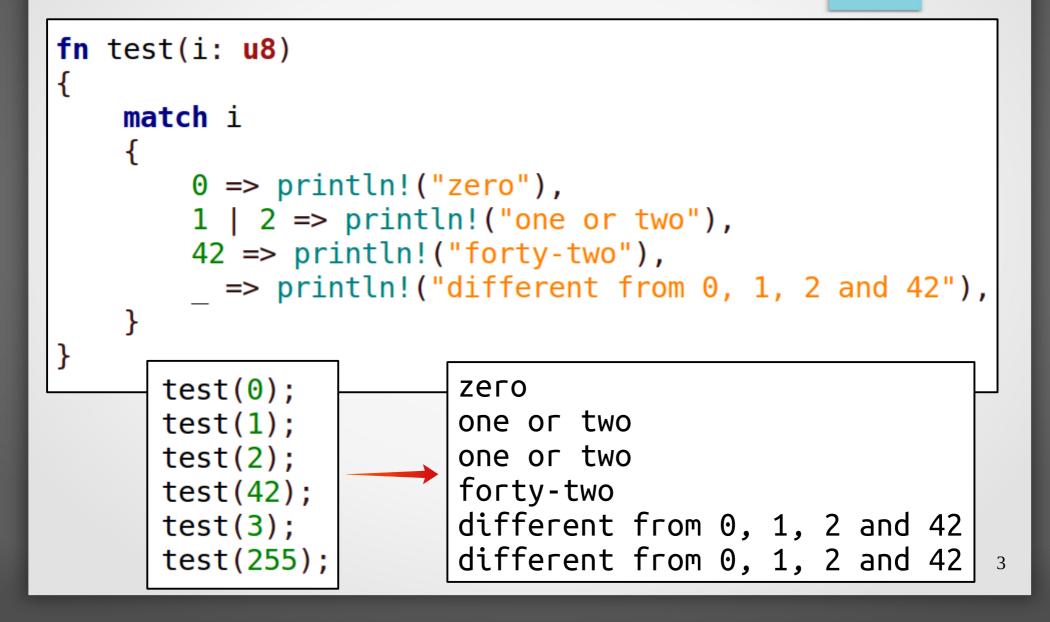
david.bouchet.epita@gmail.com

#### The match Keyword

# The match keyword compares a value to a list of patterns.

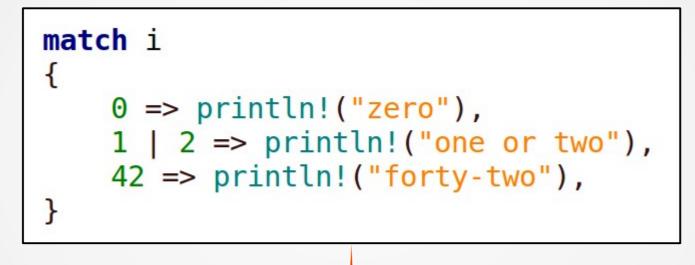
When a first pattern matches the value, the code associated to this pattern is executed and the following patterns are ignored.

#### Simple Cases



#### **Matches Are Exhaustive**

#### All cases must be tested.



# **Order Matters**

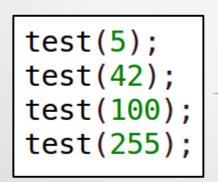
#### The match Expression

```
fn test(i: u8)
    let s = match i
    ł
        0 => "zero",
        1 | 2 => "one or two",
        42 => "forty-two",
          \Rightarrow "different from 0, 1, 2 and 42",
    };
    println!("{}", s);
       test(0);
                          zero
       test(1);
                          one or two
       test(2);
                          one or two
                          forty-two
       test(42);
                          different from 0, 1, 2 and 42
       test(3);
                          different from 0, 1, 2 and 42
       test(255);
```

6

## Matching Ranges of Integers

```
fn test(i: u8)
{
    println!("{}", match i
    {
        0 ... 9 => "lower than ten",
        10 ... 99 => "lower than one hundred",
        100 => "one hundred",
        _ => "greater than one hundred",
    });
}
```



lower than ten lower than one hundred one hundred greater than one hundred

#### Matching Ranges of Characters

test('Z');

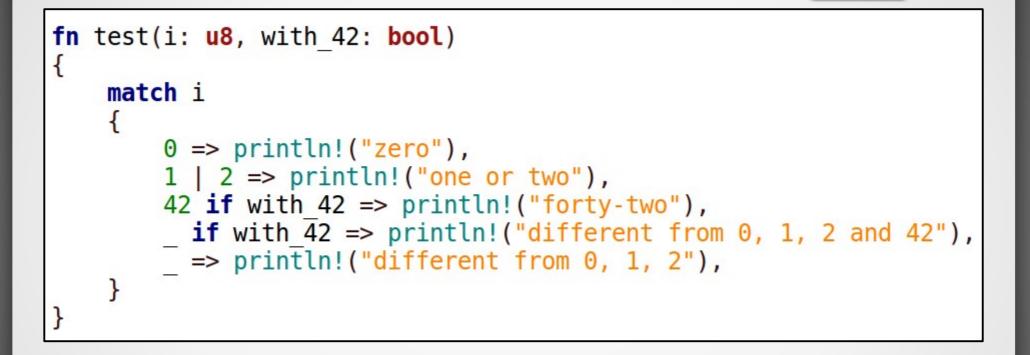
test('a');

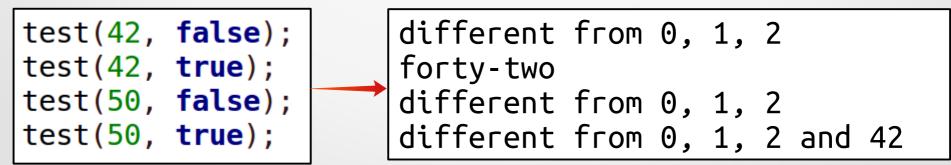
```
fn test(c: char)
{
    println!("'{}' is {}.", c, match c
    ł
        'A' ... 'M' => "between 'A' and 'M'",
        'N' \ldots 'Z' => "between 'N' and 'Z'",
         => "not a capital letter",
    });
}
                    'C' is between 'A' and 'M'.
  test('C');
```

'Z' is between 'N' and 'Z'.

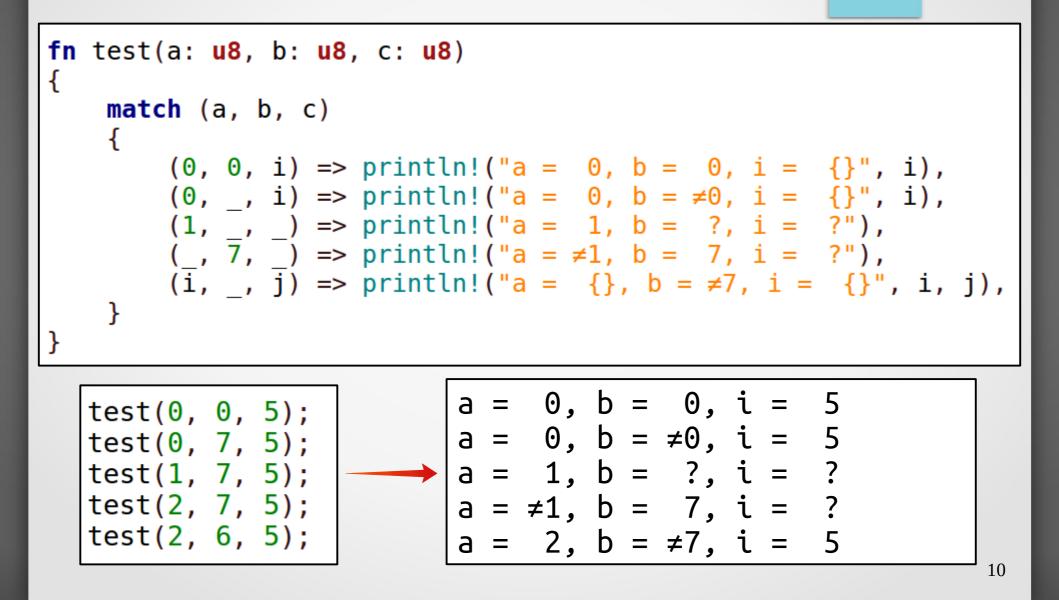
'a' is not a capital letter.

#### Match Guards

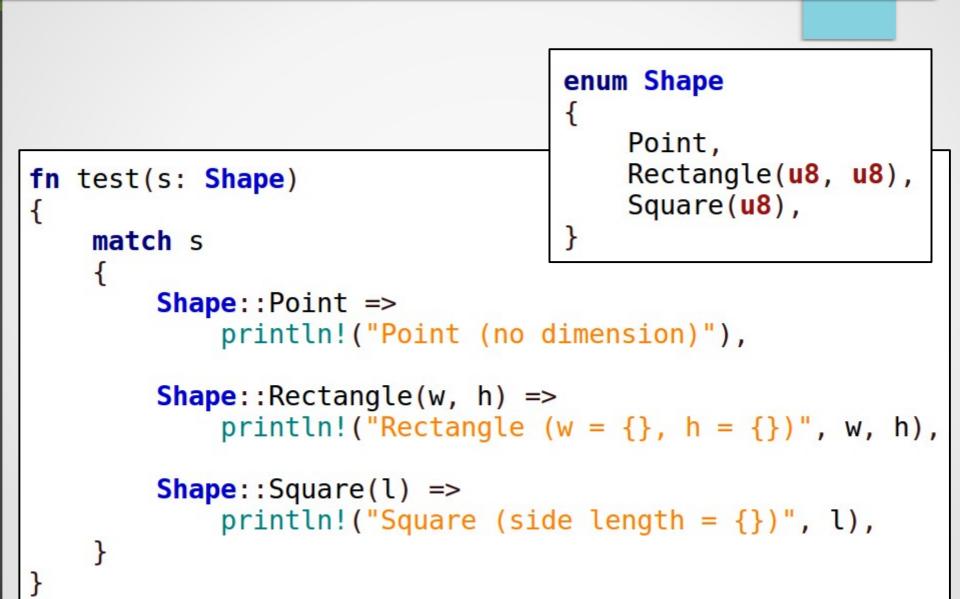




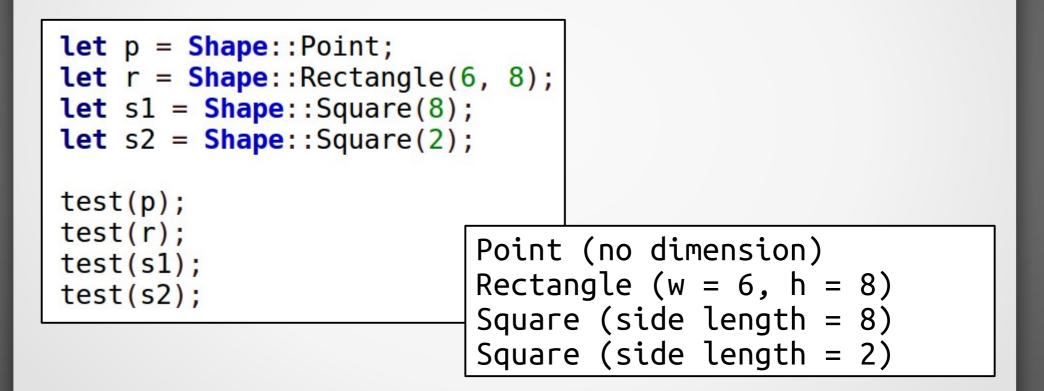
#### **Matching Tuples**



# Matching Enumerations (1)



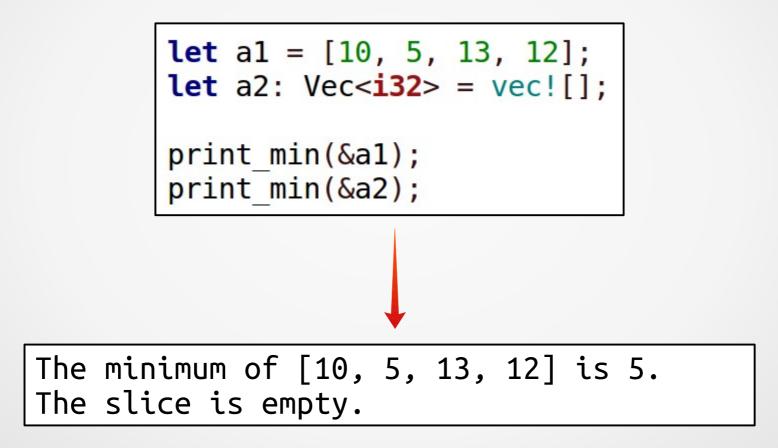
#### Matching Enumerations (2)



# Example with Option (1)

```
fn print_min(slice: &[i32])
{
    let min = slice.iter().min();
    match min
    {
        None =>
            println!("The slice is empty."),
            Some(m) =>
            println!("The minimum of {:?} is {}.", slice, m),
        }
}
```

#### Example with Option (2)



# Matching with if let (1)

```
if let Some(x) = Some(2)
{
    dbg!(x);
}
let s = Some(2);
if let Some(x) = s
{
    dbg!(x);
}
```

# Matching with if let (2)

```
fn test1(opt: Option<i32>)
    match opt
        Some(i) => println!("{}", i),
        None => println!("None"),
    }
fn test2(opt: Option<i32>)
    if let Some(i) = opt {
                              test1(Some(5));
                                                     5
        println!("{}", i);
                              test2(Some(5));
                                                     5
                              test1(None);
                                                     None
    else {
                              test2(None);
                                                     None
        println!("None");
    }
                                                             16
```

#### Matching with while let

```
let a = [10, 12, 51];
let mut iter = a.iter();
while let Some(x) = iter.next()
{
    dbg!(x);
}
```

```
x = 10
x = 12
x = 51
```

#### Error Handling with Result and ? (1)

```
fn div(a: f64, b: f64) -> Result<f64, String>
ł
    if b == 0.0
        Err(String::from("Division by zero."))
    else
        0k(a / b)
```

#### Error Handling with Result and ? (2)

```
fn print_div(a: f64, b: f64)
{
    print!("{} / {} = ", a, b);
    match div(a, b)
    {
        Err(msg) => println!("{}", msg),
        Ok(r) => println!("{}", r),
    }
}
```

print\_div(10.0, 2.0);
print\_div(3.0, 0.0);
print\_div(-10.0, 2.0);
print\_div(-10.0, -2.0);

19

#### Error Handling with Result and ? (3)

```
fn sqrt(a: f64) -> Result<f64, String>
{
    if a.is_sign_negative()
    {
        Err(String::from("Square root of negative number."))
    }
    else
    {
        Ok(a.sqrt())
    }
}
```

#### Error Handling with Result and ? (4)

```
fn print_sqrt_div(a: f64, b: f64)
{
    print!("sqrt_div({} / {}) = ", a, b);
    match sqrt_div(a, b)
    {
        Err(msg) => println!("{}", msg),
        Ok(r) => println!("{}", r),
    }
}
```

#### Error Handling with Result and ? (5)

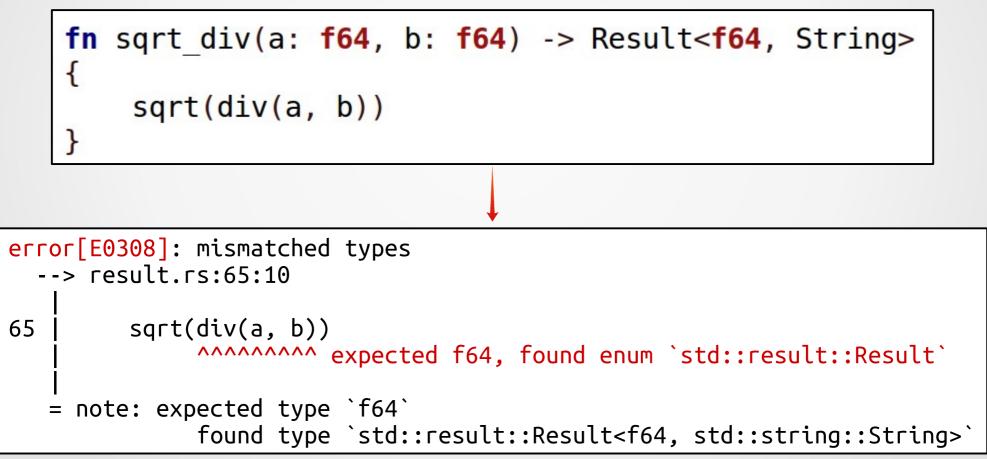
sqrt\_div(10 / 2) = 2.23606797749979
sqrt\_div(3 / 0) = Division by zero.
sqrt\_div(-10 / 2) = Square root of negative number.
sqrt\_div(-10 / -2) = 2.23606797749979

#### Error Handling with Result and ? (6)

#### Version 1

fn sqrt\_div(a: f64, b: f64) -> Result<f64, String>
{
 sqrt(div(a, b))
}

#### Error Handling with Result and ? (7)



#### Error Handling with Result and ? (8)

```
fn sqrt_div(a: f64, b: f64) -> Result<f64, String>
{
    match div(a, b)
    {
        Err(msg) => Err(msg),
        Ok(q) => sqrt(q),
    }
}
```

#### Error Handling with Result and ? (9)

```
fn sqrt_div(a: f64, b: f64) -> Result<f64, String>
{
    let r = div(a, b);
    if let 0k(q) = r { sqrt(q) } else { r }
}
```

#### Error Handling with Result and ? (10)

```
fn sqrt_div(a: f64, b: f64) -> Result<f64, String>
{
    let r = div(a, b)?;
    sqrt(r)
}
```

#### Error Handling with Result and ? (11)

#### Version 5

fn sqrt\_div(a: f64, b: f64) -> Result<f64, String>
{
 sqrt(div(a, b)?)
}