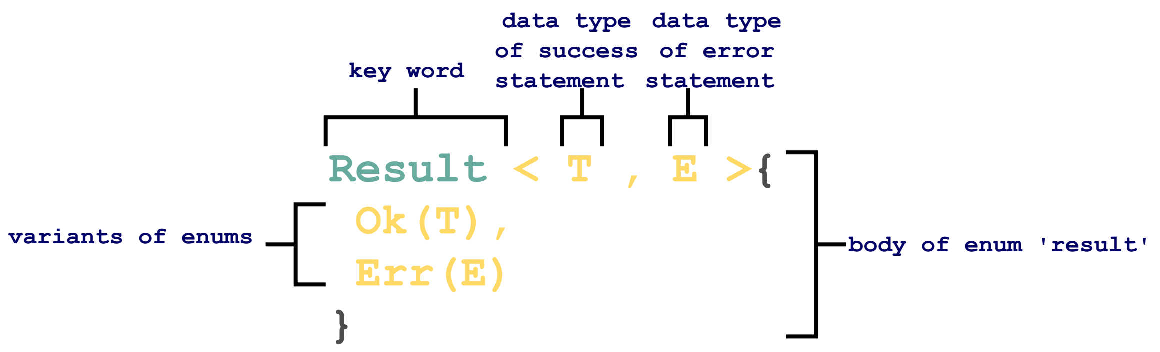
Result and Enum

This lesson will teach you about a built-in enum called result.

We'll cover the following

What Is Result?

Result is a **built-in** enum in the Rust standard library. It has two variants `Ok(T)` and `Err(E)`.



Defining a result enum

Variants:

variants:

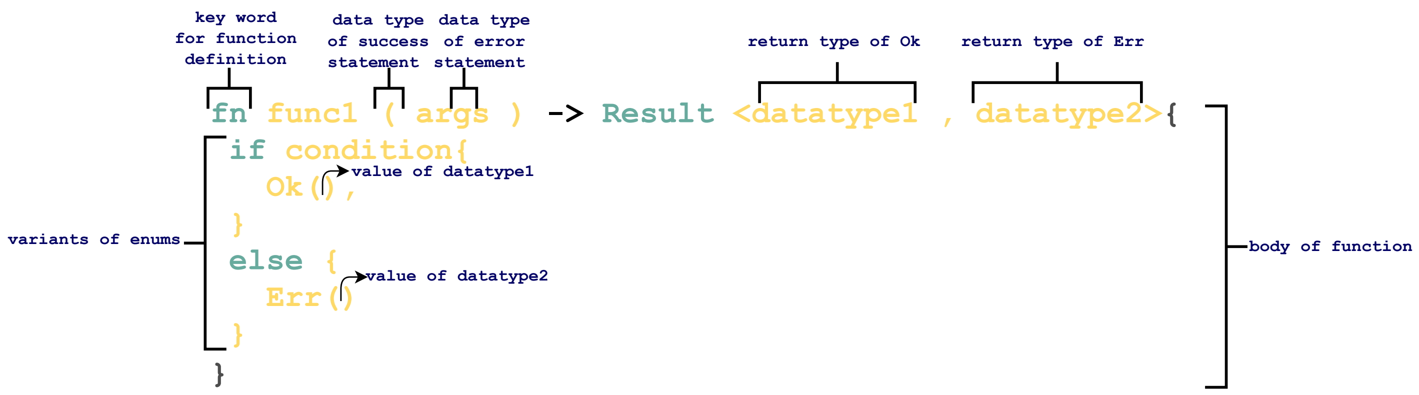
- `Ok(T)`, returns the success statement of type `T`
- `Err`, returns the error statement of type `E`.

When to Use Result?

`Result` should be used as a return type for a function that can **encounter error situations**. Such functions can return an Ok value in case of success or an Err value in case of an error.

Result and Function

Using Result as a function return type can be used to return various kinds of success and error codes to let the calling function decode the execution state of the called function.


key word for function definition
data type of success statement
data type of error statement
return type of Ok
return type of Err
fn func1 (args) -> Result <datatype1 , datatype2>{
if condition{
Ok ( ) ,
value of datatype1
}
else {
Err ( )
value of datatype2
}
}
variants of enums
body of function

Defining a result enum

Example 1

The following code has a function `file_found` which takes a number `i` and returns a `Result` of type `i32`, in case of variant `Ok` and `bool`, in case of `Err`.

```
fn main() {
   println!("{:?}",file_found(true)); // invoke function by passing true
   println!("{:?}",file_found(false)); // invoke function by passing false
}
fn file_found(i:bool) -> Result<i32,bool> {
   if i { // if true
      Ok(200) // return Ok(200)
   } else { // if false
      Err(false) // return Err(false)
   }
}
```

Explanation

- The `main` function is defined from **line 1 to 4**.
  - On **line 2**, a function `file_found` is invoked with passing a boolean value `true` as a parameter to the function and it's return value is printed.
  - On **line 3**, a function `file_found` is invoked with passing a boolean value `false` as a parameter to the function and it's return value is printed.
- From **line 5 to line** 7, the function `file_found` is defined. The function takes a boolean value `i` and returns a value of type `Result` with `Ok` variant datatype `i32` and `Err` variant datatype `bool`.
  - On **line 6-7**, `if` statement takes the argument `i` and returns `Ok(200)` if the value of `i` is `true`.
  - On **line 8-9**, `else` returns `Err(false)` if the value of `i` is `false`.

Example 2

The following code has a function `divisible_by_3` which takes a number `i` and returns a `Result` of type `String` in case of both variants `Ok` and `Err`. If `i` is divisible by 3 `Ok(Given number is divisible by 3)` is returned and `Err(Given number is not divisible by 3)`.

```
fn main() {
   println!("{:?}", divisible_by_3(6)); // invoke function by passing a number 6
   println!("{:?}", divisible_by_3(2)); // invoke function by passing a number 2
}
fn divisible_by_3(i:i32)->Result<String,String> {
   if i % 3 == 0 { // if number mod 3 equals 0
      Ok("Given number is divisible by 3".to_string()) // return this statement
   } else { // if if number mod 3 is not equals 0
      Err("Given number is not divisible by 3".to_string()) // return this statement
   }
}
```

Explanation

- The `main` function is defined from **line 1 to 4**.
  - On **line 2**, a function `divisible_by_3` is invoked with passing an integer value `6` as a parameter to the function and it's return value is printed.
  - On **line 3**, a function `divisible_by_3` is invoked with passing an integer value `2` as a parameter to the function and it's return value is printed.
- From **line 5 to line 11**, the function `divisible_by_3` is defined The function

- From **line 5 to line 11**, the function `divisible_by_3` is defined.The function takes an integer value `i` and returns a `Result` with `Ok` and `Err` variants both

  have datatype `String`.
    - On **line 6**, `if` the condition `i % 3 == 0` evaluates to true, then it returns a string on **line** 7 saying that the number is divisible by 3.
    - On **line 8**, `else` construct executes if the `if` does not execute. It returns a string on **line 9** saying that the number is not divisible by 3.

`is_ok()`, `is_err()` Functions

Rust helps you to check whether the variable of type `Result` is set to `Ok` or `Err`.

Example 1

The following example checks if the above example 2 returns an `ok` value or `err` using the `is_ok()` and `is_err()`.

```
fn main() {
   let check1 = divisible_by_3(6);
   if check1.is_ok(){ // check if the function returns ok
      println!("The number is divisible by 3");
   }
   else{
      println!("The number is not divisible by 3");

   }
   let check2 = divisible_by_3(2);
   if check2.is_err(){ // check if the function returns error
      println!("The number is not divisible by 3");
   }
   else{
      println!("The number is divisible by 3");

   }
}
fn divisible_by_3(i:i32)->Result<String,String> {
   if i % 3 == 0 { // check i modulus 3
      Ok("Given number is divisible by 3".to_string())
   } else {
      Err("Given number is not divisible by 3".to_string())
   }
}
```

Explanation

- The `main` function is defined from **line 1 to 18**.
    - On **line 2**, a variable `check1` is assigned the returned value of the

- On **line 2**, a variable check1 is assigned the returned value of the function `divisible_by_3`.
- On **lines 3-9**, an `if..else` construct checks if the function returns `ok` using `is_ok()` method. If it does, it prints that the number is divisible by 3.
- On **line 10**, a variable `check2` is assigned the returned value of the function `divisible_by_3`.
- On **lines 11-17**, an `if..else` construct checks if the function returns `err` using `is_err()` method. If it does, it prints that the number is not divisible by 3.

- The function `divisble_by_3` is defined from **line 19 to line 25**.

> The explanation of this function is given in Example 2.

To ensure that these functions return true or false, use `assert_eq` and `assert_ne`.

Example 2

The following example uses the `assert_eq!` macro to check whether the variable value of type `Result` is set to `Ok` or `Err`.

```
fn main() {
   let check1 = divisible_by_3(6);
   assert_eq!(check1.is_ok(), true);  // left is true and right is true so the assertion passes
   let check2 = divisible_by_3(2);
   assert_eq!(check2.is_err(), true); // left is true and right is true so the assertion passes
}
fn divisible_by_3(i:i32)->Result<String,String> {
   if i % 3 == 0 {
      Ok("Given number is divisible by 3".to_string())
   } else {
      Err("Given number is not divisible by 3".to_string())
   }
}
```

Explanation

- On **line 2**, a variable `check1` is assigned the return value of the function `divisible_by_3`.

- On **line 3**, An assert_eq! takes the expression `check.is_ok` and checks if it’s equal to `true`.
- On **line 4**, a variable `check2` is assigned the return value of the function `divisible_by_3`.
- On **line 5**, An assert_eq! takes the expression `check2.is_err` and checks if it’s equal to `false`.

> **Note:** The assertion passes since the expression evaluates to true.

Now that you have learned about enums, let’s check your knowledge in the upcoming challenge.