# Pass by Reference

In this lesson, passing arguments by reference will be introduced to you.

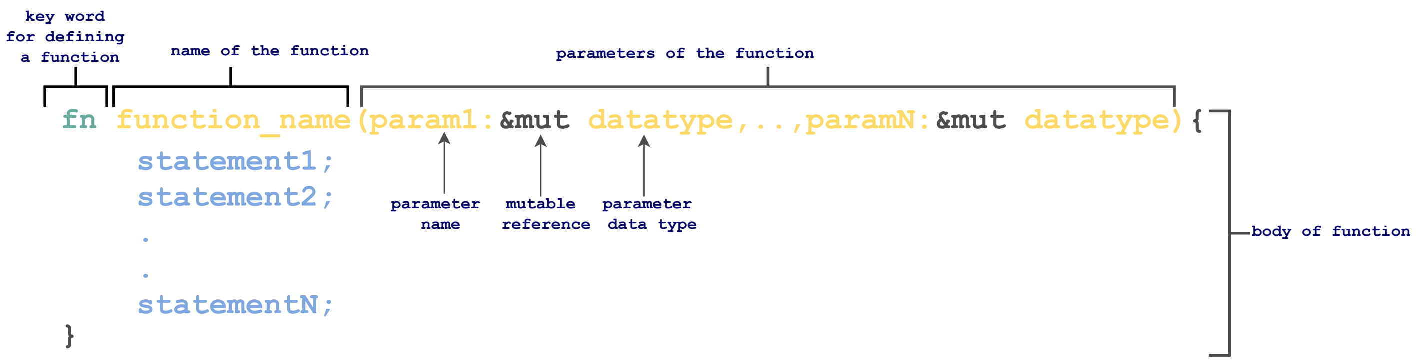
**We'll cover the following**

## Arguments Pass by Reference #

When we want the called function to make changes to the parameters such that the changes are seen by the calling function when the call returns. The mechanism to do this is called pass arguments by reference.

### Syntax #

The general syntax of passing arguments by value is:



```
fn function_name(param1:&mut datatype,..,paramN:&mut datatype){
    statement1;
    statement2;
    .
    .
    statementN;
}
```

Defining a function with values passed by reference

## Example #

The following example makes a function `square()` that takes a number `n` which is being passed by reference as a parameter to the function and prints the square of

the function within the function.

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}", n);
}
fn main() {
  let  mut n = 4;
  println!("The value of n before function call : {}", n);
  println!("Invoke Function");
  square(&mut n);
  println!("The value of n after function call : {}", n);
}
```

## Explanation

The above program comprises two functions, the user defined function `square()` and the driver function `main()` where the function is being called.

### User defined function

The function `square()` is defined from **line 1 to line 4** which takes a mutable reference (`&mut`) to the parameter `n` of type `i32`.

- On *line 2*, the square of the variable `n` is calculated. Since `n` is a reference to a variable, to access the referenced variable's value, a de-referencing is required. That is achieved with the `*n`. On the right handside, the value referenced by `n` is accessed and multiplied with itself. The assignment is also to `*n`, which means the calculated result is stored in the variable that `n` is referencing.

- The square of the function is printed on *line 3*.

### Driver function

The driver function `main()` is defined from **line 5 to line 11**.

- On *line 6*, a mutable variable `n` is defined.

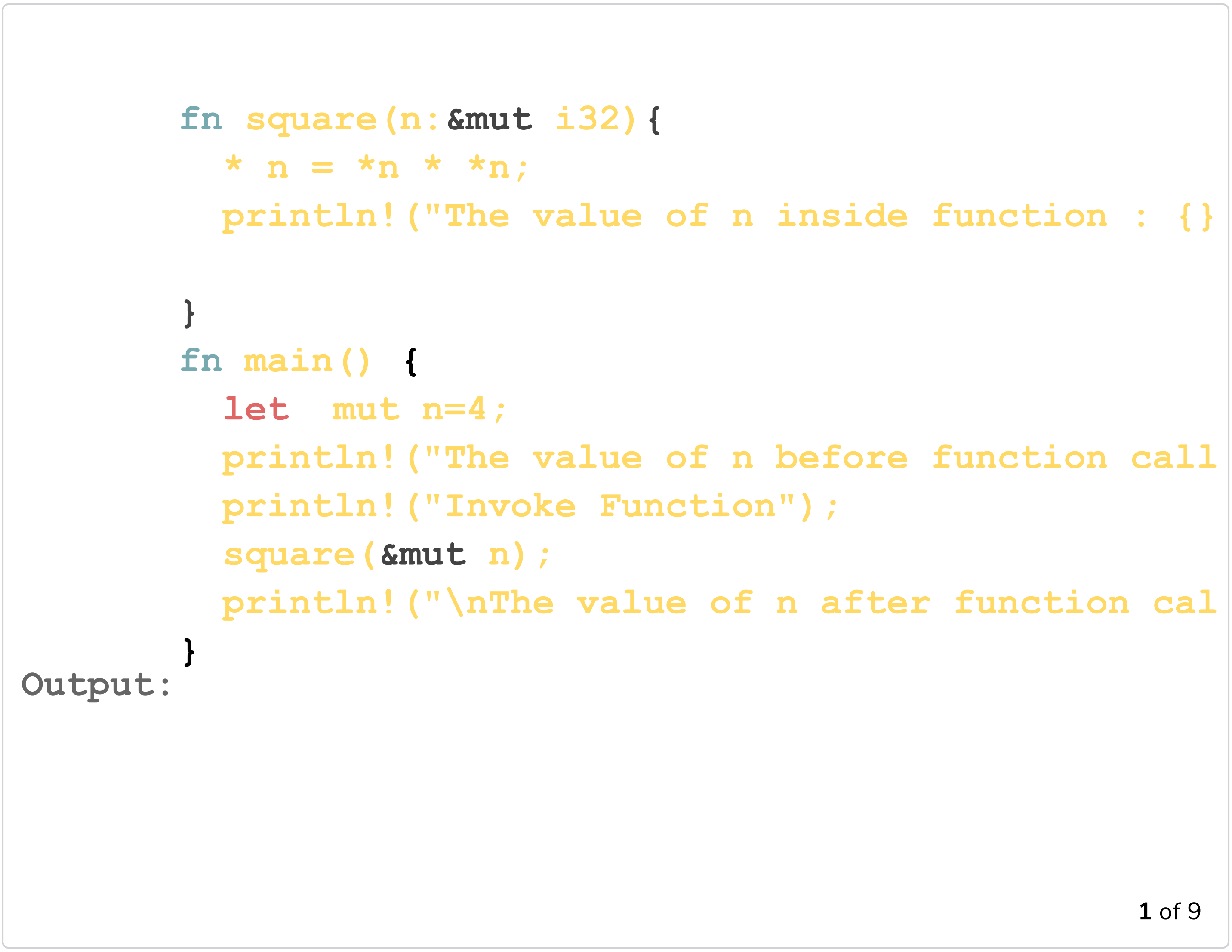
- On *line 9*, the function `square()` is invoked.The argument to this function is `&mut n`. Here, `&` indicates that it is a reference to the variable `n` and `mut` indicates that `n` can be changed inside the function `square()`.

- After the function call, the value of the `n` is printed.

> **Note:** The value of `n` **is changed** within the function.

> **Note:** The argument, as well as the parameter, is set as a mutable reference when the value is passed by reference. If the value is to be updated it is dereferenced first and then the update operation is performed.

The following illustration shows how program execution proceeds in the above code:



```
fn square(n:&mut i32){
  * n = *n * *n;
  println!("The value of n inside function : {}

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n);
  println!("\nThe value of n after function cal
}
```

```
Output:
```

1 of 9

```
        fn square(n:&mut i32){
          * n = *n * *n;
          println!("The value of n inside function : {}

        }
        fn main() {
          let  mut n=4;
          println!("The value of n before function call
          println!("Invoke Function");
          square(&mut n);
          println!("\nThe value of n after function cal
        }
Output:
```

```
        fn square(n:&mut i32){
          * n = *n * *n;
          println!("The value of n inside function : {}

        }
        fn main() {
          let  mut n=4;
          println!("The value of n before function call
          println!("Invoke Function");
          square(&mut n);
          println!("\nThe value of n after function cal
        }
Output:The value of n before the function call: 4
```

```
fn square(n:&mut i32){
  * n = *n * *n;
  println!("The value of n inside function : {}

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function cal
}
```

```
Output:The value of n before the function call: 4
       Invoke Function
```

n=4

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}"

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function call
}
```

```
Output:The value of n before the function call: 4
       Invoke Function
```

n=4

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}"

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function call
}
```

```
Output:The value of n before the function call: 4
       Invoke Function
```

n=4

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}"

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function call
}
```

```
Output:The value of n before the function call: 4
       Invoke Function
       The value of n inside function : 16
```

n=4

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}"

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function call
}
```

```
Output:The value of n before the function call: 4
       Invoke Function
       The value of n inside function : 16
       The value of n after function call :16
```

```
fn square(n:&mut i32){
  *n = *n * *n;
  println!("The value of n inside function : {}"

}
fn main() {
  let  mut n=4;
  println!("The value of n before function call
  println!("Invoke Function");
  square(&mut n); Mutable reference to n
  println!("\nThe value of n after function call
}end of program code
```

```
Output:The value of n before the function call: 4
       Invoke Function
       The value of n inside function : 16
       The value of n after function call :16
```

## Quiz #

Test your understanding of passing arguments by reference in a function in Rust.

Quick Quiz on Pass by Reference!

Q

What is the output of the following code?

```
fn change(x:&mut i32, y:&mut i32){
  *x = 0;
  *y = 0;
  println!("x : {}, y : {}", x , y);
}
fn main() {
  let mut x = 10;
  let mut y = 9;
  change( &mut x, &mut y );
  println!("x : {}, y : {}", x , y);
}
```

Retake Quiz

Now that you have learned how to pass values by reference, in the next lesson you’ll learn to return values from a function.