

Scope or Lifetime?

same, same but different

Scope

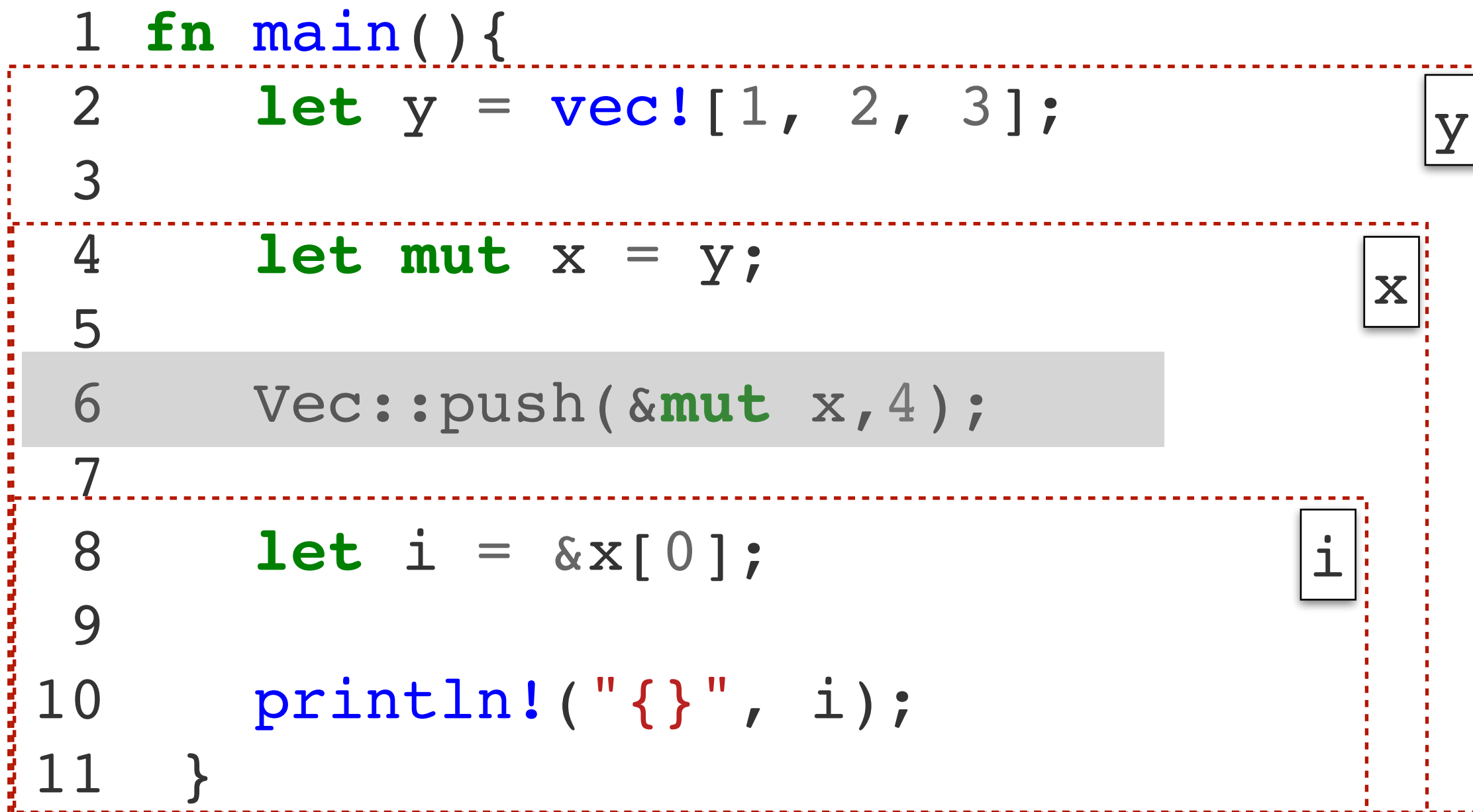
```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     x.push(4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```

A **scope** indicates the code block where a **variable is valid**:

- starts where the variable is first introduced,
- ends at the corresponding closing “ } ”.

Scope

```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     Vec::push(&mut x, 4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```

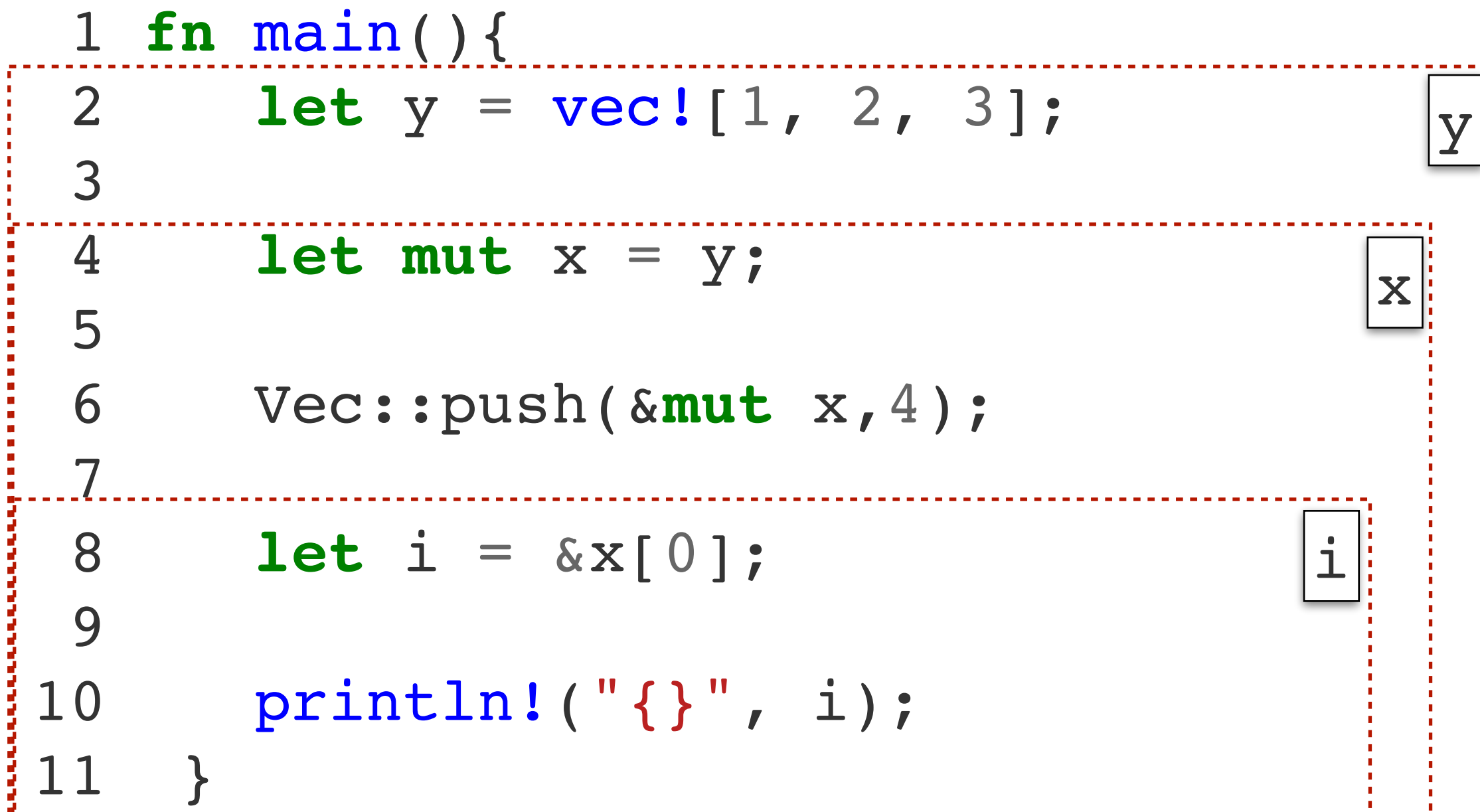


A **scope** indicates the code block where a **variable is valid**:

- starts where the variable is first introduced,
- ends at the corresponding closing “ } ”.

Scope

```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     Vec::push(&mut x, 4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```

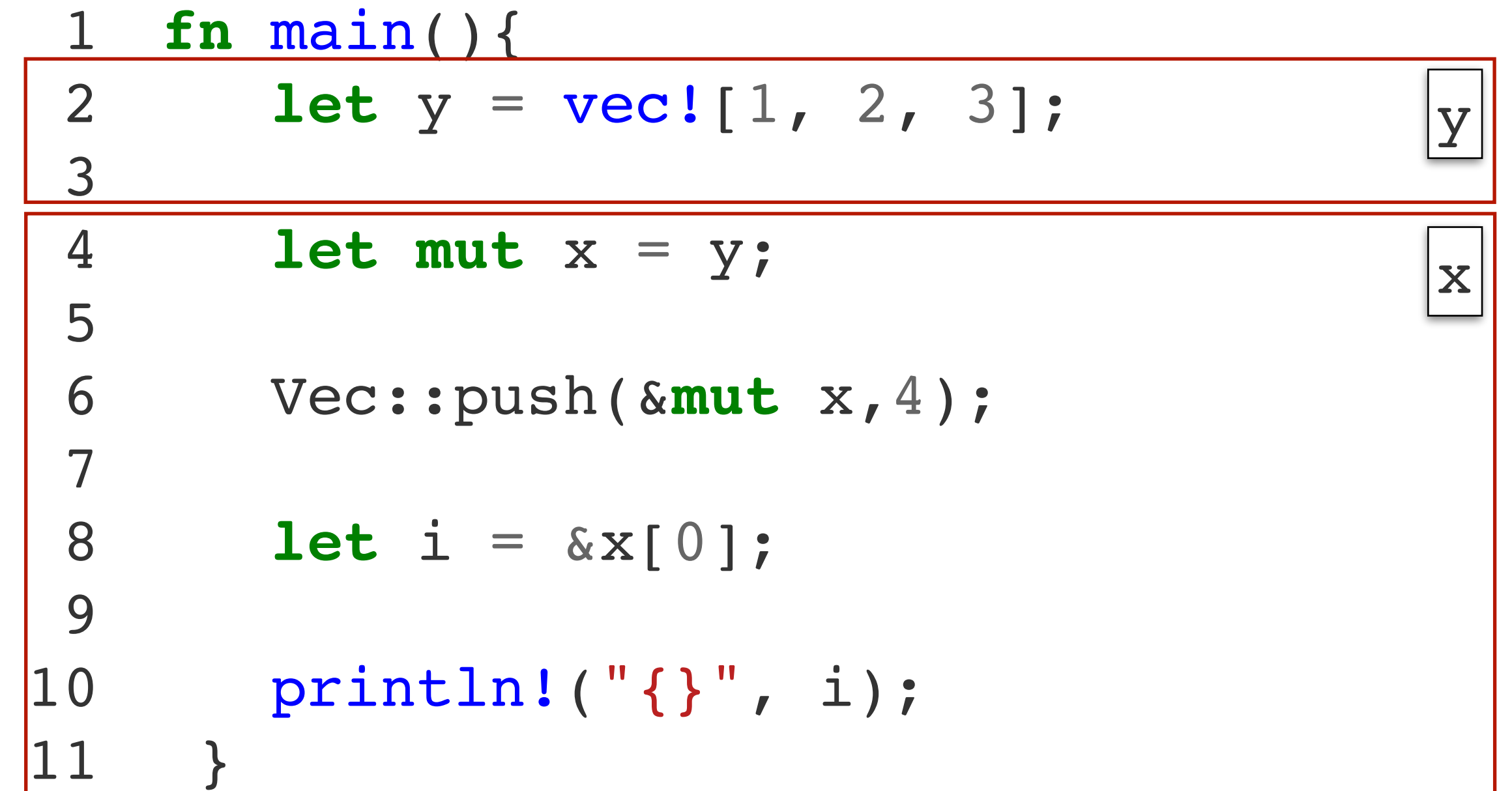


A **scope** indicates the code block where a **variable is valid**:

- starts where the variable is first introduced,
- ends at the corresponding closing “ } ”.

Liveness

```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     Vec::push(&mut x, 4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```



A **variable is live (owner is valid)** from initialisation until:

- its value is moved, or
- it goes out of scope (and is dropped).

Liveness

Safety Principle 1

Every value has a variable that's called its owner.
There can only be one owner at a time.

```
1  fn main() {  
2      let y = vec![1, 2, 3];  
3  
4      let mut x = y;  
5  
6      Vec::push(&mut x, 4);  
7  
8      let i = &x[0];  
9  
10     println!("{}", i);  
11 }
```

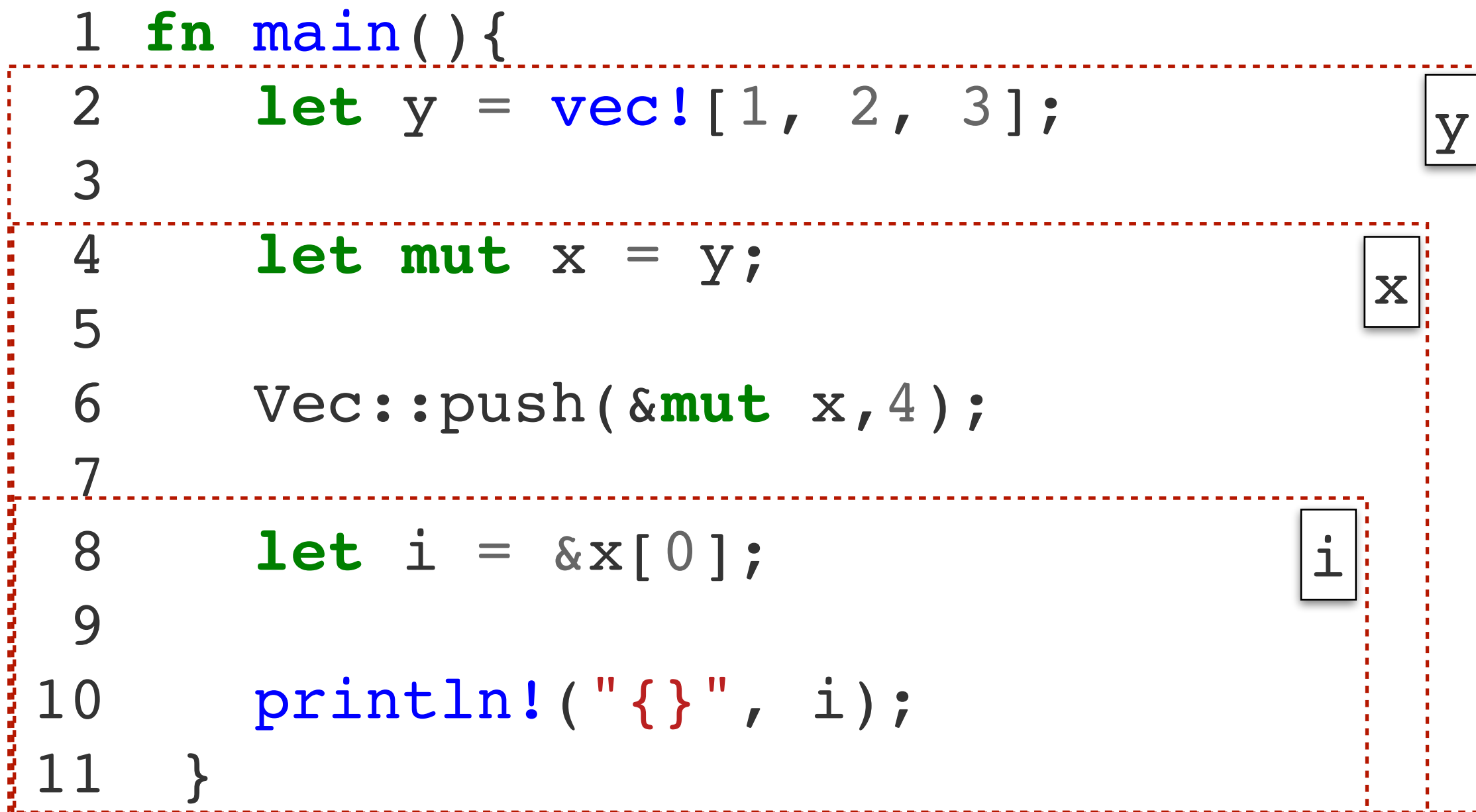
y

x

- A **variable is live (owner is valid)** from initialisation until:
- its value is moved, or
 - it goes out of scope (and is dropped).

Scope

```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     Vec::push(&mut x, 4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```

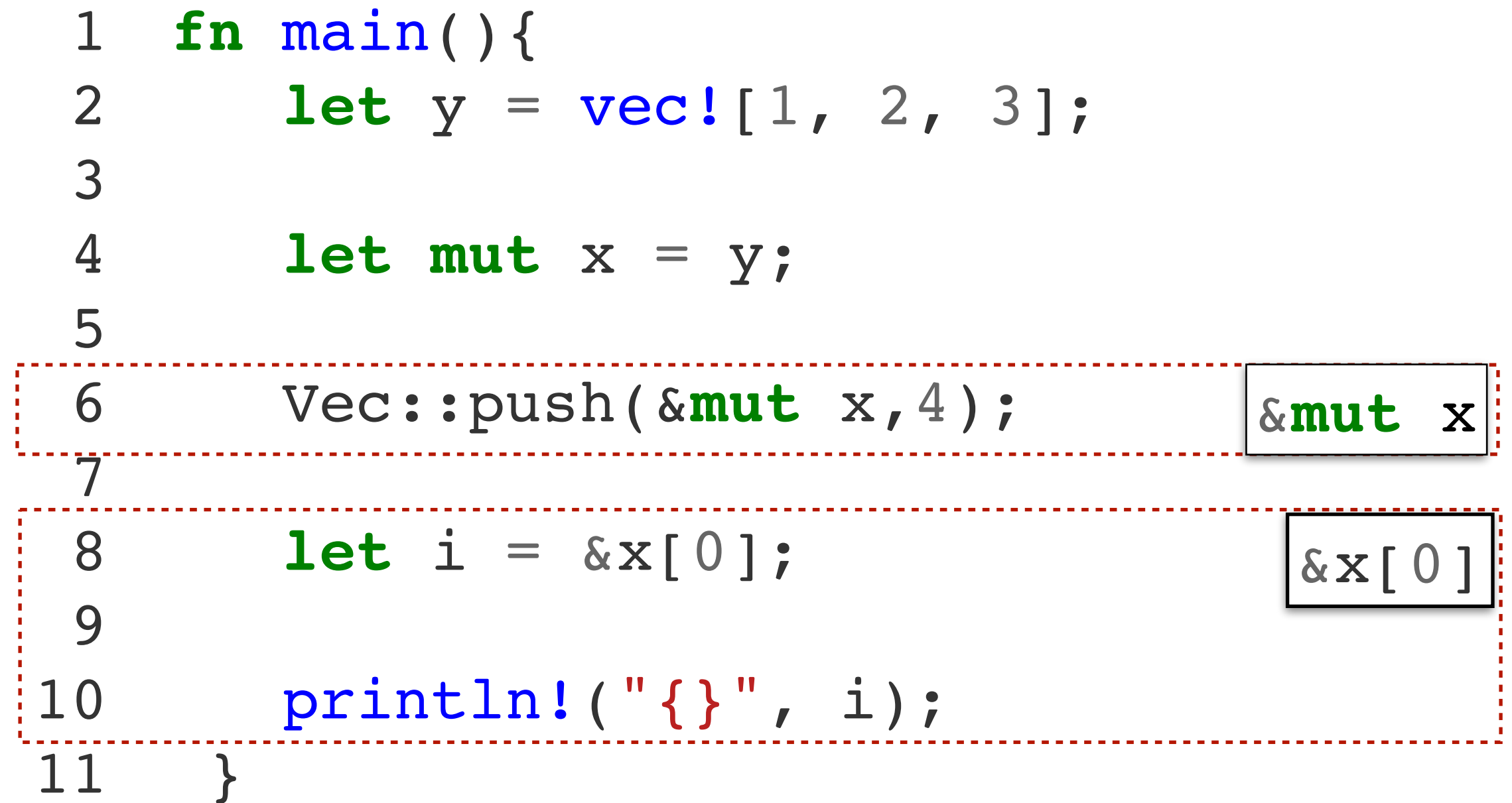


A **scope** indicates the code block where a **variable is valid**:

- starts where the variable is first introduced,
- ends at the corresponding closing “ } ”.

Non-Lexical Lifetime

```
1 fn main() {  
2     let y = vec![1, 2, 3];  
3  
4     let mut x = y;  
5  
6     Vec::push(&mut x, 4);  
7  
8     let i = &x[0];  
9  
10    println!("{}", i);  
11 }
```



A **lifetime** indicates the code block where a **borrow is valid**:

- starts where the reference is created,
- ends where the reference is last used/needed.

Non-Lexical Lifetime

Safety Principle 3

A mutable borrow can only be created if its lender has no other borrows living at that time.

```
1  fn main() {  
2      let y = vec![1, 2, 3];  
3  
4      let mut x = y;  
5  
6      Vec::push(&mut x, 4);  
7  
8      let i = &x[0];  
9  
10     println!("{}", i);  
11 }
```

The diagram shows two overlapping regions of code, each enclosed in a red dashed box. The first region, labeled 'b' in a yellow box, covers lines 6 and 7. The second region, labeled 'a' in a yellow box, covers lines 8, 9, and 10. The first region contains the code `Vec::push(&mut x, 4);` and the second region contains `let i = &x[0];` and `println!("{}", i);`. The labels 'b' and 'a' are placed in yellow boxes to the right of the first and second regions, respectively. The text `&mut x` is shown in a box to the right of line 6, and `&x[0]` is shown in a box to the right of line 8.

A **lifetime** indicates the code block where a **borrow is valid**:

- starts where the reference is created,
- ends where the reference is last used/needed.

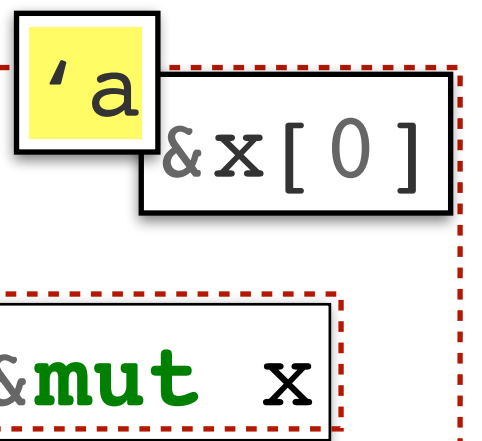
Non-Lexical Lifetime

Safety Principle 3

A mutable borrow can only be created if its lender has no other borrows living at that time.



```
1  fn main() {  
2      let y = vec![1, 2, 3];  
3  
4      let mut x = y;  
5  
6      let i = &x[0];  
7  
8      Vec::push(&mut x, 4);  
9  
10     println!("{}", i);  
11 }
```



A **lifetime** indicates the code block where a **borrow is valid**:

- starts where the reference is created,
- ends where the reference is last used/needed.

Non-Lexical Lifetime

Safety Principle 3

A mutable borrow can only be created if its lender has no other borrows living at that time.



```
1  fn main() {  
2      let y = vec![1, 2, 3];  
3  
4      let mut x = y;  
5  
6      let i = &x[0];  
7  
8      Vec::push(&mut x, 4);  
9  
10     println!("{}", i);  
11 }
```

'a

'b

A **lifetime** indicates the code block where a **borrow is valid**:

- starts where the reference is created,
- ends where the reference is last used/needed.

Safety Principle 4

The lender cannot be modified as long as one of its (shared) borrowers still lives.



```
1 fn main() {  
2     let mut x = vec![1, 2, 3];  
3  
4     let i = &x;  
5  
6     x = vec![3, 4];  
7  
8     println!("{}", i[0]);  
9 }
```

'a

Safety Principle 2

The lender needs to outlive
all of its (alive) references.



```
1 fn main() {  
2     let mut x = vec![1, 2, 3];  
3  
4     let i = &x;  
5  
6     let y = x;  
7  
8     println!("{}", i[0]);  
9 }
```

x

'a'

Reborrowing

Safety Principle 1

Every value has a variable that's called its owner.
There can only be one owner at a time.

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

Safety Principle 3

A mutable borrow can only be created if its
lender has no other borrows living at that time.

Safety Principle 4

The lender cannot be modified as long as
one of its (shared) borrowers still lives.

```
1 fn main() {  
2     let mut x = vec![10, 11];  
3  
4     let v = &mut x;  
5  
6     let i = &mut (*v)[0];  
7  
8     println!("x[0] = {}", *i);  
9  
10    Vec::push(v, 12);  
11 }
```

```

1  fn main() {
2      let v = vec![10, 20, 30];
3      let def = 0;
4      let ref_d = &def;
5
6
7      let r: &i32 =
8          if v.len()>0 { &v[0] }
9          else { ref_d } ;
10
11     println!("{}", r);
12 }

```

'b

'a

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

A **lifetime** indicates the code block where a **borrow** is valid:

- starts where the reference is created,
- ends where the reference is last used/needed.

```

1  fn main() {
2      let v = vec![10, 20, 30];
3      let def = 0;
4      let ref_d = &def;
5
6
7      let r: &i32 =
8          if v.len()>0 { &v[0] }
9          else { ref_d } ;
10
11     println!("{}", r);
12 }

```

'b

'a

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

A **lifetime** indicates the code block where a **borrow** is valid:

- starts where the reference is created,
- ends where the reference is last used/needed.

```
1  fn main() {
2      let v = vec![10, 20, 30];
3      let def = 0;
4      let ref_d = &def;
5
6
7      let r: &i32 =
8          if v.len()>0 { &v[0] }
9          else { ref_d } ;
10
11     println!("{}", r);
12 }
```

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

```
if v.len()>0 { &v[0] }  
else { ref_d } ;
```

```
1  fn main() {  
2      let v = vec![10, 20, 30];  
3      let def = 0;  
4      let ref_d = &def;  
5  
7      let r: &i32 =  
8  
9  
10  
11      println!("{}", r);  
12 }
```

Safety Principle 2

The lender needs to outlive
all of its (alive) references.


```
fn get_first(v: &Vec<i32>, ref_d: &i32) -> &i32 {  
    if v.len()>0 { &v[0] }  
    else { ref_d } ;  
}
```

```
1 fn main() {  
2     let v = vec![10, 20, 30];  
3     let def = 0;  
4     let ref_d = &def;  
5  
7     let r: &i32 =  
8  
9  
10  
11     println!("{}", r);  
12 }
```

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

```
fn get_first(v: &Vec<i32>, ref_d: &i32) -> &i32 {  
    if v.len()>0 { &v[0] }  
    else { ref_d } ;  
}
```

```
1  fn main() {  
2      let v = vec![10, 20, 30];  
3      let def = 0;  
4      let ref_d = &def;  
5  
7      let r: &i32 = get_first(&v, ref_d);  
8  
9  
10  
11      println!("{}", r);  
12 }
```

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

Non-Lexical Lifetime

(parameterised)

```
fn get_first<'a>(v: &'a Vec<i32>, ref_d:&'ai32) -> &'ai32 {  
    if v.len()>0 { &v[0] }  
    else { ref_d } ;  
}
```

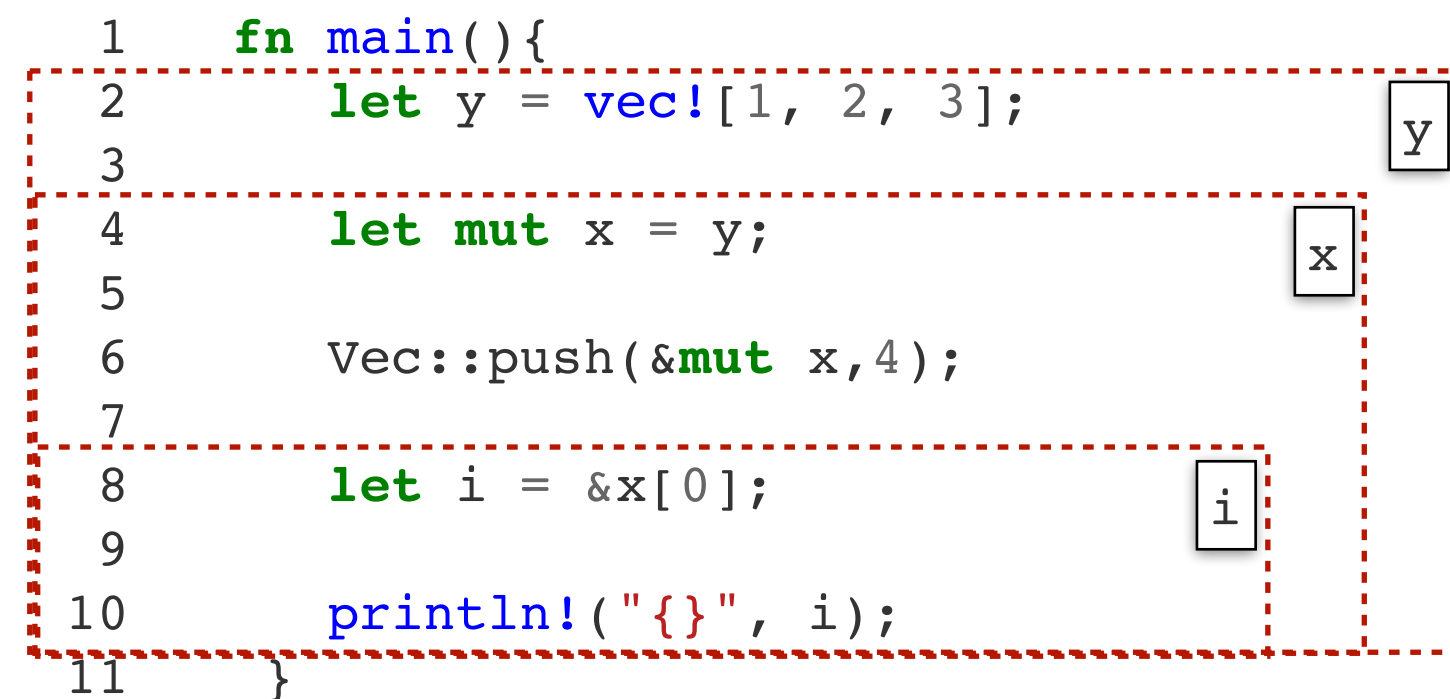
```
1  fn main() {  
2      let v = vec![10, 20, 30];  
3      let def = 0;  
4      let ref_d = &def;  
5  
7      let r: &i32 = get_first(&v, ref_d);  
8  
9  
10  
11      println!("{}", r);  
12 }
```

Safety Principle 2

The lender needs to outlive
all of its (alive) references.

Scope

```
1  fn main(){
2      let y = vec![1, 2, 3];
3
4      let mut x = y;
5
6      Vec::push(&mut x, 4);
7
8      let i = &x[0];
9
10     println!("{}", i);
11 }
```

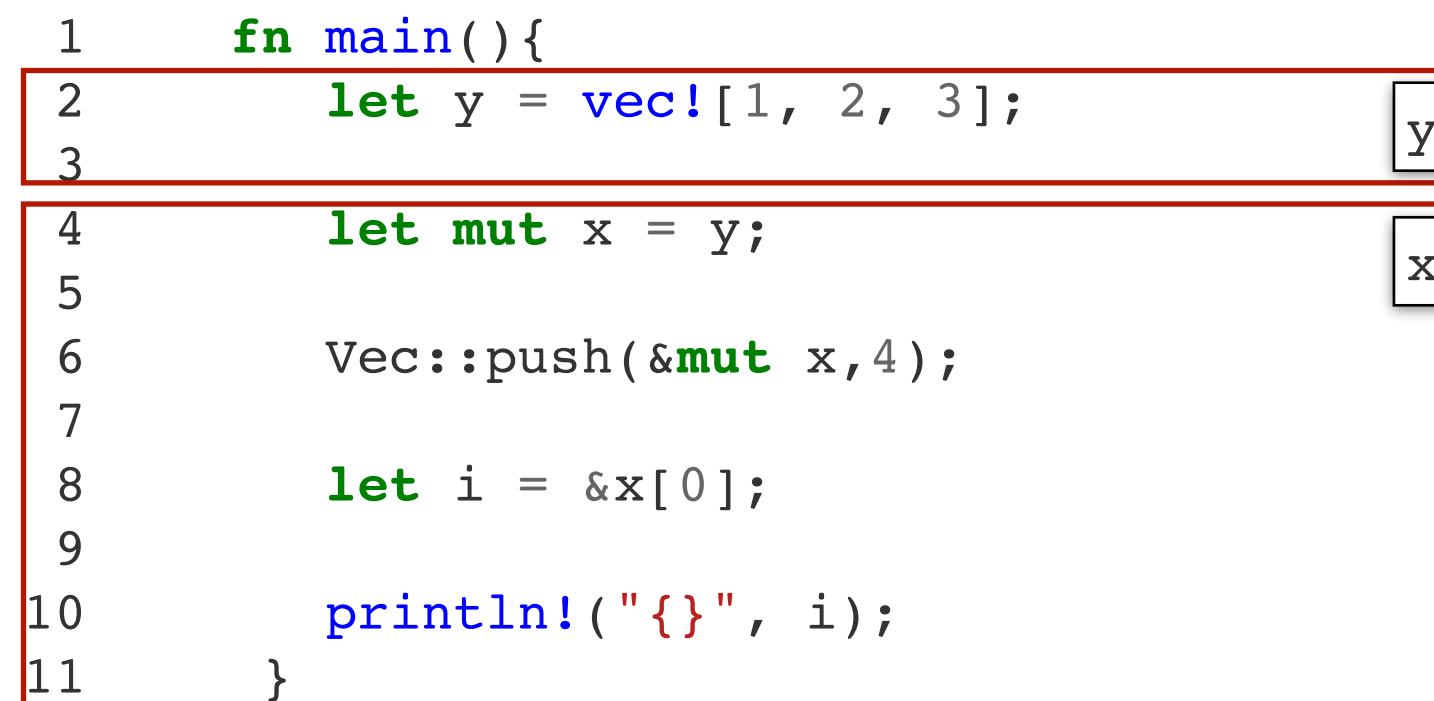


A **scope** indicates the code block where a **variable is valid**:

- starts where the variable is first introduced,
- ends at the corresponding closing “}”.

Liveness

```
1  fn main(){
2      let y = vec![1, 2, 3];
3
4      let mut x = y;
5
6      Vec::push(&mut x, 4);
7
8      let i = &x[0];
9
10     println!("{}", i);
11 }
```

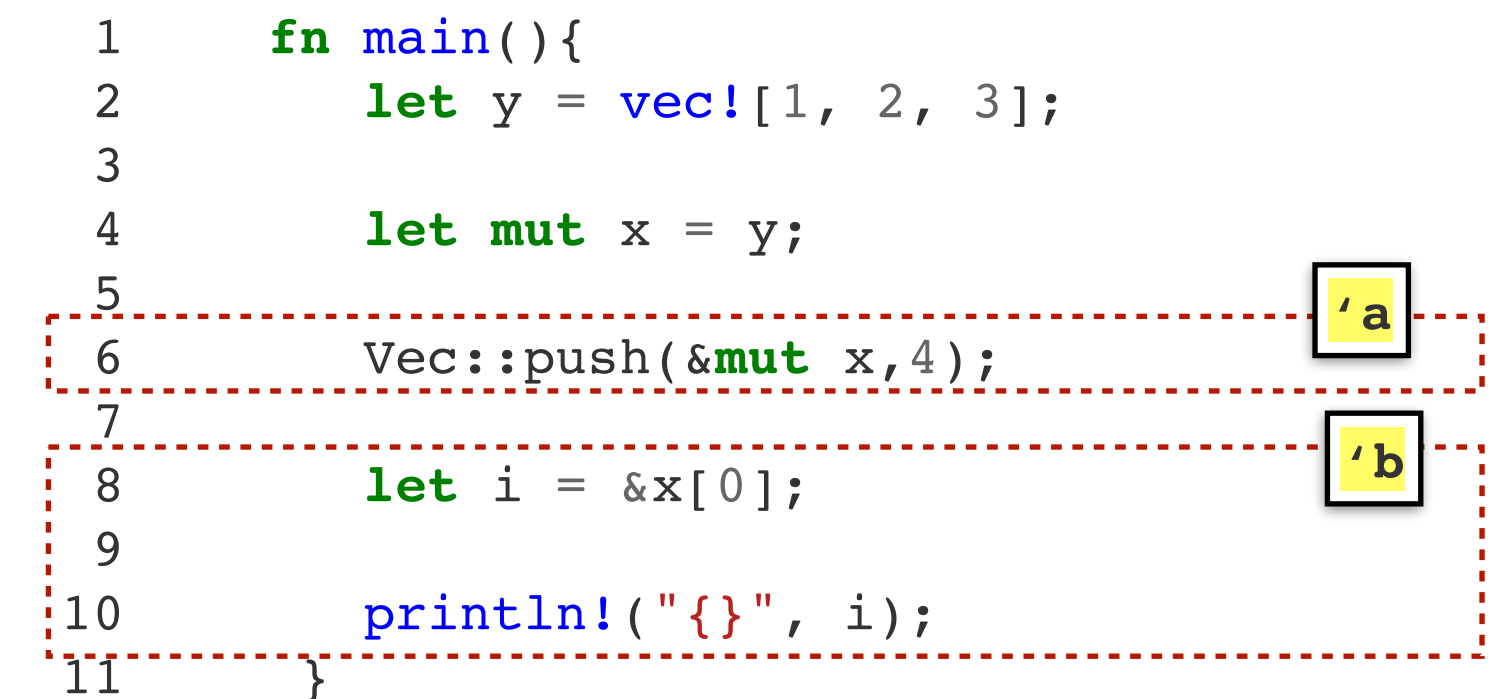


A **variable is live (owner is valid)** from initialisation until:

- its value is moved, or
- it goes out of scope (and is dropped).

Non-Lexical Lifetime

```
1  fn main(){
2      let y = vec![1, 2, 3];
3
4      let mut x = y;
5
6      Vec::push(&mut x, 4);
7
8      let i = &x[0];
9
10     println!("{}", i);
11 }
```



A **lifetime** indicates the code block where a **borrow is valid**:

- starts where the reference is created,
- ends where the reference is last used/needed.

<https://cel.cs.brown.edu/aquascope>

