**Practical Programming** 

### **Rust : Arrays and Vectors**



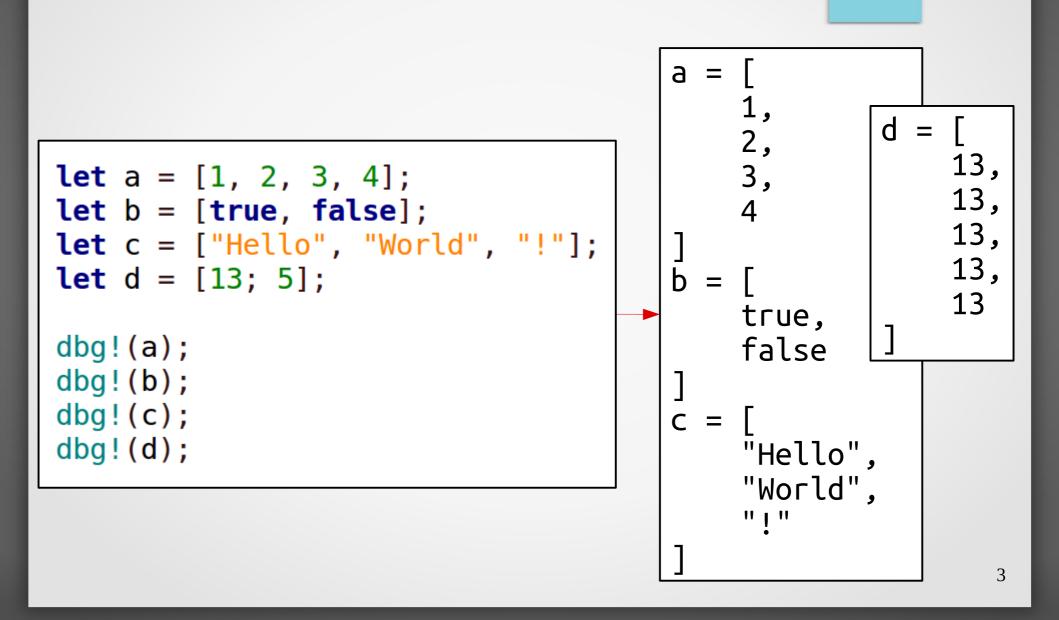
#### **David Bouchet**

david.bouchet.epita@gmail.com

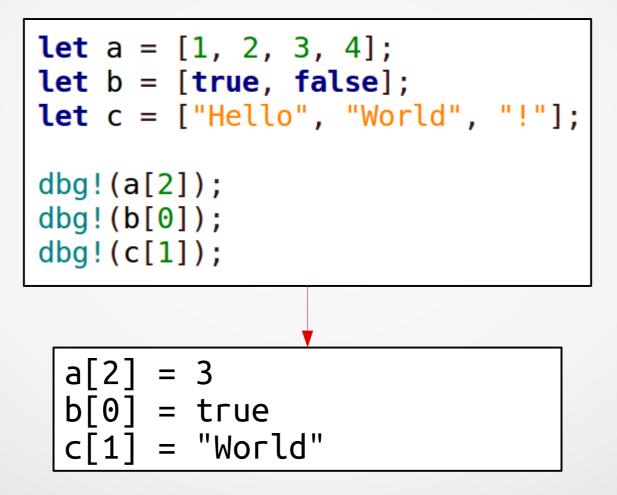
#### Arrays

- Every element has the same type.
- The length is fixed.
- Data is allocated on the stack.
- Elements cannot be added or removed.
- If mutable, elements can change.

#### **Creating Arrays**

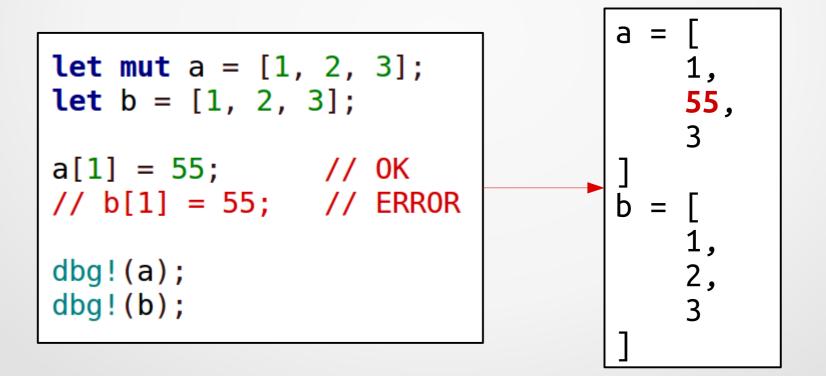


#### **Indexing Arrays**

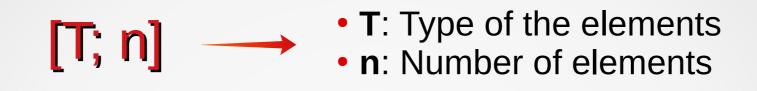


Mutable and Immutable Arrays

Elements cannot be added or removed. Elements can change in mutable arrays only.



#### Array Types



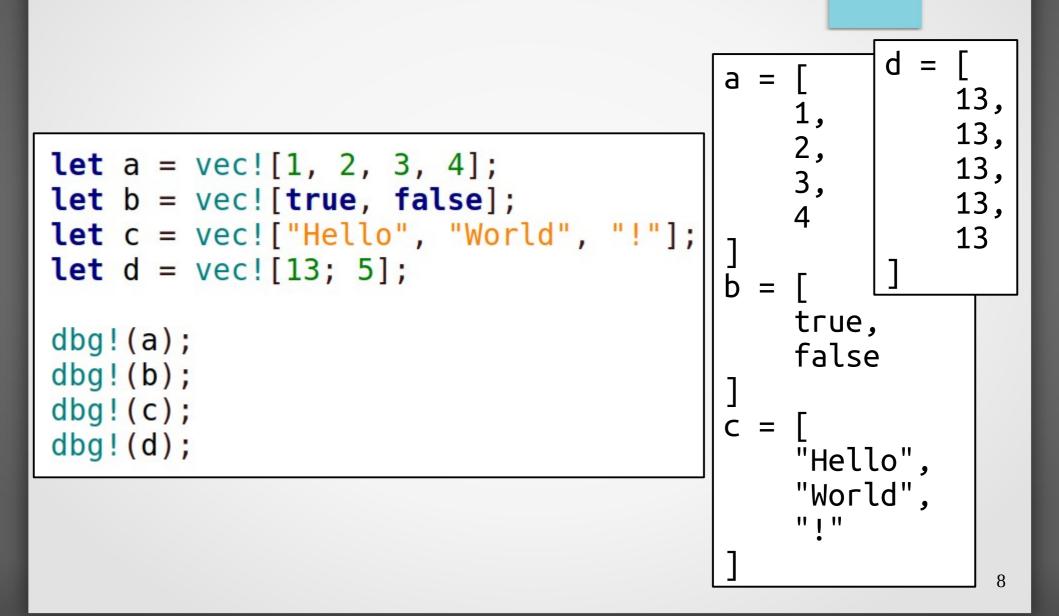
let mut e = [1, 2, 3]; // [i32; 3] e = [2, 3, 4]; // [i32; 3] e = [5, 6, 7, 8]; // [i32; 4] -> ERROR

The length belongs to the type.

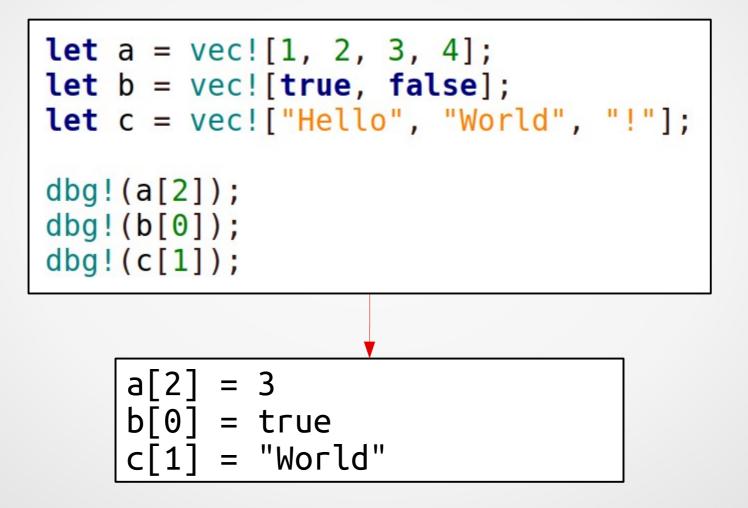
#### Vectors

- Every element has the same type.
- The length is variable.
- Data is allocated on the heap.
- If mutable, elements can be added or removed.
- If mutable, elements can change.

#### **Creating Vectors**



#### **Indexing Vectors**



#### **Mutable Vectors**

#### Elements can be added or removed. Elements can change

let mut v = vec!["Hello", "World", "!"]; v.insert(0, "Good"); v[1] = "bye"; v.remove(3); v.push("!!!!!"); v.insert(2, ","); dbg!(v);
V = [ "Good", "bye", ",", "World", "!!!!!"

#### **Vector Types**

```
Vec<T> T: Type of the elements
```

```
let a = vec![1, 2, 3];  // Vec<i32>
let b: Vec<u8> = vec![1, 2, 3]; // Vec<u8>
let c = vec![1_u8, 2, 3];  // Vec<u8>
let x: u8 = 8;
let d = vec![x, 2, 3];  // Vec<u8>
let mut e = vec![1, 2, 3];  // Vec<i32>
e = vec![2, 3, 4];  // Vec<i32>
e = vec![5, 6, 7, 8];  // Vec<i32> -> OK
```



#### Array and Vector Lengths

dbg!(a.len());
dbg!(v.len());

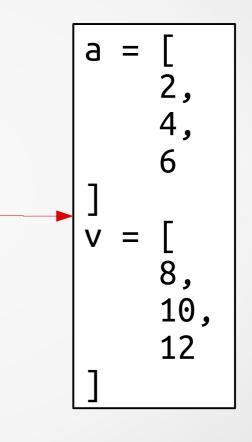
#### Iterating over Immutable References

```
let a = [1, 2, 3];
let v = vec![4, 5, 6];
for i in a.iter()
    dbg!(i);
for i in v.iter()
    dbg!(i);
                       4
                      = 5
                        6
```

- The *i* variable is an **immutable reference**.
- By default, Rust prints the value that is pointed to by the reference.
- dbg!(\*i) would give the same result.

#### Iterating over Mutable References

```
let mut a = [1, 2, 3];
let mut v = vec![4, 5, 6];
for i in a.iter mut()
{
   *i *= 2;
}
for i in v.iter mut()
{
   *i *= 2;
}
dbg!(a);
dbg!(v);
```



#### The *i* variable is a **mutable reference**.

**Printing Arrays and Vectors** 

# There is no default formatter for arrays and vectors.

```
fn main()
{
    let a: [u8; 3] = [1, 2, 3];
    println!("{}", a);
}
```

#### **Debug** Printing

To print arrays and vectors with *println!()*, we must use the **debug formatter**:

### Implicit Dereferencing

<pre>let a = [1, 2, 3]; let v = vec![4, 5, 6];</pre>	
<pre>let ref_a = &amp;a let ref_v = &amp;v</pre>	a[1] = 2 v[1] = 5 (*ref_a)[1] = (*ref_v)[1] = ref_a[1] = 2 ref_v[1] = 5
<pre>dbg!(a[1]); dbg!(v[1]);</pre>	
<pre>dbg!((*ref_a)[1]); dbg!((*ref_v)[1]);</pre>	
<pre>dbg!(ref_a[1]); dbg!(ref_v[1]);</pre>	

#### **Slice Reference**

```
\&[T] \longrightarrow T: Type of the elements
```

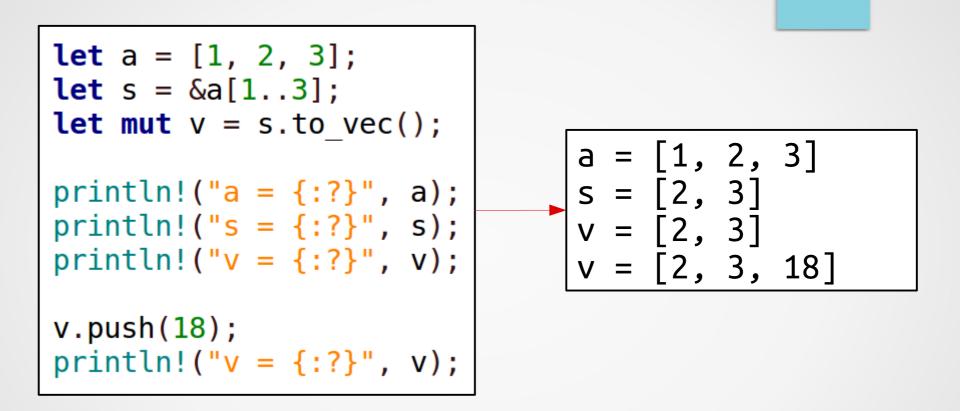
```
fn average(slice: &[i32]) -> f64
ł
    let mut sum = 0;
                               let a = [1, 2, 3];
                               let v = vec![4, 5, 6];
    for i in slice.iter()
                               dbg!(average(&a));
        sum += i;
                               dbg!(average(&v));
    }
    sum as f64 / slice.len() as f64
}
                            average(\&a) = 2.0
                            average(\&v) = 5.0
```

#### Slicing

# Array and vector slices can be referenced in the same way as string slices.

<pre>let a = [1, 2, 3, 4, 5]; let v = vec![6, 7, 8, 9, 10];</pre>		
<pre>dbg!(average(&amp;a[1]));</pre>	<pre>average(&amp;a[1]) = 3.5</pre>	
dbg!(average(&v[2]));	average(&v[2]) = 6.5	
dbg!(average(&a[1=3]));	average(&a[1=3]) = 3.0	
dbg!(average(&v[24]));	average(&v[24]) = 8.5	

#### **Converting Slices into Vectors**



- The slice is copied into another memory space, which is associated with a vector.
- In the example, s and v are independent.

#### **Mutable Slices**

**&mut**[T] — T: Type of the elements

**let mut** a = [1, 2, 3, 4, 5, 6];

println!("a = {:?}", a); set\_to\_zero(&mut a[1..=4]); println!("a = {:?}", a);

fn set\_to\_zero(slice: &mut [i32])
{
 for i in slice.iter\_mut()
 {
 \*i = 0;
 }
}