

Practical Programming

Rust : Strings



David Bouchet

david.bouchet.epita@gmail.com

Two Types of Strings

Dynamic strings: **String**

- Contents **can** change.
- Characters **can** be added or removed.

String slices: **&str**

- Contents **cannot** change.
- Characters **cannot** be added or removed.

See: <https://doc.rust-lang.org/std/string/struct.String.html>
<https://doc.rust-lang.org/std/primitive.str.html>

Dynamic Strings – Examples

```
let mut s = String::from("Hello");  
  
dbg!(s.pop().unwrap());  
dbg!(&s);  
  
s.insert_str(0, "The road to ");  
s.push_str(" is paved with");  
s += " good intentions.";  
  
dbg!(&s);
```

```
s.pop().unwrap() = 'o'
```

```
&s = "Hell"
```

```
&s = "The road to Hell is paved with good intentions."
```

String Slices – Examples

```
let dyn: String = String::from("dynamic string");
let lit: &str = "string literal is of type &str";

let slice_1: &str = &dyn[0..8];
let slice_2: &str = &dyn[0..=7];
let slice_3: &str = &lit[7..=13];
let slice_4: &str = &dyn[..];
let slice_5: &str = &dyn;
```

```
slice_1 = "dynamic "
slice_2 = "dynamic "
slice_3 = "literal"
slice_4 = "dynamic string"
slice_5 = "dynamic string"
```

Creating Dynamic Strings

```
let s1 = String::new();  
let s2 = String::from("Hello");  
let s3 = "world".to_string();  
let s4 = format!("{}", {}, s2, s3);
```

```
dbg!(s1);  
dbg!(s2);  
dbg!(s3);  
dbg!(s4);
```

```
s1 = ""  
s2 = "Hello"  
s3 = "world"  
s4 = "Hello, world!"
```

Breaking String Literals (1)

```
let s = "This string  
is made up  
of  
four lines."  
println!("{}", s);
```



```
This string  
    is made up  
    of  
    four lines.
```

Breaking String Literals (2)

```
let s = "This string \  
is made up \  
of \  
one line."  
  
println!("{}", s);
```

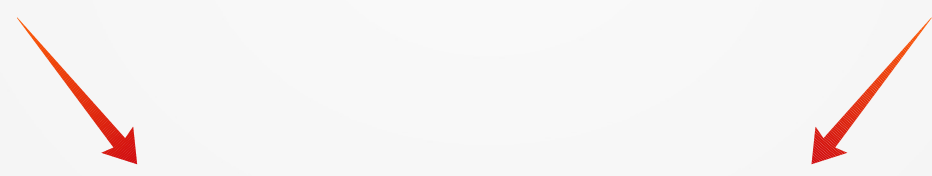


```
This string is made up of one line.
```

Breaking String Literals (3)

```
let s = "This string  
is made up  
of  
four lines."  
  
println!("{}", s);
```

```
let s = "This string\n\  
is made up\n\  
of\n\  
four lines."  
  
println!("{}", s);
```



```
This string  
is made up  
of  
four lines.
```


UTF-8 Encoding

String: é ≠ è → 5 characters

In memory:

C3 A9 20 E2 89 A0 20 C3 A8

↓
é

↓
space

↓
≠

↓
space

↓
è

→ 9 bytes

Indexing Strings (1)

What are `s[0]` and `s[8]`?

```
let s = "é ≠ è"
```

```
C3 A9 20 E2 89 A0 20 C3 A8
```

Indexing Strings (2)

What are `s[0]` and `s[8]`?

```
let s = "é ≠ è"
```

```
C3 A9 20 E2 89 A0 20 C3 A8
```

➔ They are undefined!

Indexing Strings (3)

Strings cannot be indexed.

```
fn main()
{
    let s = "Hello";
    dbg!(s[0]);
}
```



```
error[E0277]: the type `str` cannot be indexed by `{integer}`
--> string_indices.rs:4:10
```

```
4 |     dbg!(s[0]);
  |           ^^^^ `str` cannot be indexed by `{integer}`
```

Scanning Strings (1)

To scan a string, we need an iterator.

```
for var in iterator
{
    // ...
}
```

Two commonly used methods:

- `chars()`: returns an iterator over the **chars** of a string.
- `bytes()`: returns an iterator over the **bytes** of a string.

Scanning Strings (2)

Iterating over Characters

```
let s = "é ≠ è";  
for c in s.chars()  
{  
    dbg!(c);  
}
```



```
c = 'é'  
c = ' '  
c = '≠'  
c = ' '  
c = 'è'
```

Scanning Strings (3)

Iterating over Bytes

```
let s = "é ≠ è";  
print!("bytes: ");  
for b in s.bytes()  
{  
    print!("{:x} ", b);  
}
```

bytes: c3 a9 20 e2 89 a0 20 c3 a8

Scanning Strings (4)

Other useful iterators are available

Two examples:

- `lines()`: returns an iterator over the **lines** of a string.
- `split_whitespace()`: returns an iterator over sub-slices that are separated by any amount of whitespace.

Scanning Strings (5)

Iterating over Lines

```
let s = "Performance\n\nReliability\n\nProductivity";  
  
for line in s.lines()  
{  
    dbg!(line);  
}
```


→

```
line = "Performance"  
line = "Reliability"  
line = "Productivity"
```

Scanning Strings (6)

Iterating over Words

```
let s = "Rust is blazingly fast";  
for word in s.split_whitespace()  
{  
    dbg!(word);  
}
```



```
word = "Rust"  
word = "is"  
word = "blazingly"  
word = "fast"
```

String Lengths

```
let s = "é ≠ è";
```

```
let length_in_characters = s.chars().count();
```

```
let length_in_bytes = s.len();
```

```
dbg!(length_in_characters);
```

```
dbg!(length_in_bytes);
```



```
length_in_characters = 5  
length_in_bytes = 9
```

The *next()* Method

```
let s = "Rust";  
let mut iter = s.chars();  
  
dbg!(iter.next().unwrap());  
dbg!(iter.next().unwrap());  
dbg!(iter.next().unwrap());  
dbg!(iter.next().unwrap());
```

Why *mut*?

Why *unwrap()*?

```
iter.next().unwrap() = 'R'  
iter.next().unwrap() = 'u'  
iter.next().unwrap() = 's'  
iter.next().unwrap() = 't'
```

The *next()* Method – Why *mut*?

Each time the *next()* method is called, the iterator **must change an inner value** that points to the next item to return.

To be allowed to change this value, it has to be mutable.

The *next()* Method – Why *unwrap()*?

The *next()* method does not return the item directly. It returns a **special type** that contains the item.

This special type is an **enumeration**. We have not looked at enumerations yet.

We will see how to remove the *unwrap()* method in a further lesson.

The *unwrap()* Method

It **extracts** the item from the enumeration and **returns** it.

If no item can be extracted, this method **panics**.

When a **panic** occurs, the program prints a failure message, cleans up the stack and exits.

Be careful, functions that panic should be used during the design process only.

Panic – Example

```
let s = "Rust";  
let mut iter = s.chars();  
  
dbg!(iter.next().unwrap()); // 'R'  
dbg!(iter.next().unwrap()); // 'u'  
dbg!(iter.next().unwrap()); // 's'  
dbg!(iter.next().unwrap()); // 't'  
dbg!(iter.next().unwrap()); // Panics!!!
```



```
iter.next().unwrap() = 'R'  
iter.next().unwrap() = 'u'  
iter.next().unwrap() = 's'  
iter.next().unwrap() = 't'  
thread 'main' panicked at 'called  
`Option::unwrap()` on a `None` value'
```


Foretaste – Without *unwrap()*

```
let s = "Rust";  
let mut iter = s.chars();
```

```
dbg!(iter.next());  
dbg!(iter.next());  
dbg!(iter.next());  
dbg!(iter.next());  
dbg!(iter.next());  
dbg!(iter.next());
```

```
iter.next() = Some('R')  
iter.next() = Some('u')  
iter.next() = Some('s')  
iter.next() = Some('t')  
iter.next() = None  
iter.next() = None
```