Strings in rust

String - The Permanent Sticky Note

- This is like a full sheet of paper where you can write, erase, and add more
- You own it completely - it's yours to modify
- It lives in your notebook (the heap memory)

Technically

- Growable, mutable, owned UTF-8 encoded string type
- Stored on the heap
- You can modify it (add/remove characters)

- Example: Your personal to-do list that you keep updating

```rust
let mut my_note = String::from("Buy milk");
my_note.push_str(" and eggs"); // You can add more
println!("{}", my_note); // Prints "Buy milk and eggs"
```

&str - The Borrowed Sticky Note

- This is like a small sticky note someone lets you look at
- You can't change it - it's fixed
- It might be pointing to part of someone else's permanent note or a pre-written message

Technically

- Immutable reference to a string (either a String or a string literal)
- Fixed size, can't be modified
- Often used as function parameters to accept either string literals or `String` values
-

- Example: A store's "Open" sign that you can read but can't alter

```rust
let store_sign = "Open"; // This is a &str
let permanent_note = String::from("Open 9-5");
let borrowed_part = &permanent_note[0..4]; // Also a &str ("Open")


```

Why Two Types?

Rust does this to be super-efficient with memory:

- `String` is for when you need to build or change text
- `&str` is for when you just need to read text (faster and safer)

Important Things to Know

1. Rust strings always use proper text encoding (UTF-8)
2. You can't just grab one "letter" directly because some characters take more space (like emojis)
3. If you need to change a string, you must use `String` and declare it as `mut` (mutable)

Methods

1. push_str - method takes a string slice
2. push - method takes a single character as a parameter and adds it to the `String`

Indexing in strings

- Rust strings don't support indexing
- the number of bytes it takes to encode "Здравствуйте" in UTF-8, because each Unicode scalar value in that string takes 2 bytes of storage. Therefore, an index into the string's bytes will not always correlate to a valid Unicode scalar value

Rust

```rust
fn main() {


    // will this code run or not ?
   /*

    let mut s1 = String::from("foo");
    let s2 = "bar";
    s1.push_str(s2);
    println!("s2 is {s2}");

    */

    let hello = String::from("Hola");
    println!("length is {}", hello.len());

    let hello = String::from("Здравствуйте");
    println!("length is {}", hello.len());

    let hello = "Здравствуйте";
    let s = &hello[0..4];
    println!("length is {}", s);

    for c in "Зд".chars() {
        println!("{c}");
    }

    for b in "Зд".bytes() {
        println!("{b}");
    }

}


```

Key Differences

	Feature	String	&str
1	Ownership	Owns the data	Borrows the data
2	Mutability	Can be mutable	Always immutable

3	Storage	Heap	Heap or binary
4	Size	Growable	Fixed length