Table of Contents

String Package Documentation

This package provides comprehensive string manipulation utilities for MoonBit, i
ncluding string creation, conversion, searching, and Unicode handling.

String Creation and Conversion

Create strings from various sources:

```

test "string creation" {

  let chars = ['H', 'e', 'l', 'l', 'o']
  let str1 = String::from_array(chars)
  inspect(str1, content="Hello")


  let str2 = String::from_iter(['W', 'o', 'r', 'l', 'd'].iter())
  inspect(str2, content="World")


  let empty = String::default()
  inspect(empty, content="")
}
```

String Iteration

Iterate over Unicode characters in strings:

```

test "string iteration" {
  let text = "Hello??


  let chars = text.iter().collect()
  inspect(chars, content="[’H’, ’e’, ’l’, ’l’, ’o’, ’??’]"


  let reversed = text.rev_iter().collect()
  inspect(reversed, content="[’??’, ’o’, ’l’, ’l’, ’e’, ’H’]"


  let mut count = 0
  let mut first_char = ’a’
  text
  .iter2()
  .each(fn(idx, char) {
    if idx == 0 {
      first_char = char
    }
    count = count + 1
  })
  inspect(first_char, content="H")
  inspect(count, content="6")
}
```

String Conversion

Convert strings to other formats:

```

test "string conversion" {
  let text = "Hello"


  let chars = text.to_array()
  inspect(chars, content="[’H’, ’e’, ’l’, ’l’, ’o’]")


  let bytes = text.to_bytes()
  inspect(bytes.length(), content="10")
}
```

Unicode Handling

Work with Unicode characters and surrogate pairs:

```
test "unicode handling" {
  let emoji_text = "Hello??World


  let char_count = emoji_text.iter().count()
  let code_unit_count = emoji_text.length()
  inspect(char_count, content="11")
  inspect(code_unit_count, content="12")


  let offset = emoji_text.offset_of_nth_char(5)
  inspect(offset, content="Some(5)")


  let has_11_chars = emoji_text.char_length_eq(11)
  inspect(has_11_chars, content="true")
}
```

String Comparison

Strings are ordered using shortlex order by Unicode code points:

```
test "string comparison" {
  let result1 = "apple".compare("banana")
  inspect(result1, content="-1")
  let result2 = "hello".compare("hello")
  inspect(result2, content="0")
  let result3 = "zebra".compare("apple")
  inspect(result3, content="1")
}
```

String Views

String views provide efficient substring operations without copying:

```
test "string views" {
  let text = "Hello, World!"
  let view = text[:][7:12]


  let chars = view.iter().collect()
  inspect(chars, content="['W', 'o', 'r', 'l', 'd']")


  let substring = view.to_string()
  inspect(substring, content="World")
}
```

Practical Examples

Common string manipulation tasks:

```

test "practical examples" {
  let text = "The quick brown fox"


  let words = text.split(" ").collect()
  inspect(words.length(), content="4")
  inspect(words[0].to_string(), content="The")
  inspect(words[3].to_string(), content="fox")


  let word_strings = words.map(fn(v) { v.to_string() })
  let mut result = ""
  for i, word in word_strings.iter2() {
    if i > 0 {
      result = result + "-"
    }
    result = result + word
  }
  inspect(result, content="The-quick-brown-fox")


  let upper = text[:].to_upper().to_string()
  inspect(upper, content="THE QUICK BROWN FOX")
  let lower = text[:].to_lower().to_string()
  inspect(lower, content="the quick brown fox")
}
```

Performance Notes

- Use StringBuilder or Buffer for building strings incrementally rather than repea
ted concatenation
- String views are lightweight and don’t copy the underlying data
- Unicode iteration handles surrogate pairs correctly but is slower than UTF-16 co
de unit iteration
- Character length operations (char_length_eq, char_length_ge) have O(n) compl
exity where n is the character count