scaler

1. Introduction

This crate provides a convenient `Formatter` to scale, round, and display numbers.

Scaling describes the usage of decimal or binary unit prefixes to increase readability; though no scaling and scientific notation are also supported.
Rounding can be done either to a specified magnitude or to a number of significant digits.
Separators can be freely adjusted. The group separator separates groups of digits every 3 digits before the decimal separator, while the decimal separator separates the integer and fractional parts of a number.
The sign behaviour can be set to always show the sign, only show the sign when the number is negative, or never show the sign.

2. Table of Contents

3. Usage

1. Execute `Formatter::new` to create a new `Formatter` with default settings.
2. Adjust separators, rounding, scaling, and sign behaviour as necessary using the setters.
3. Format numbers with `Formatter::format`.

3.1. `Rounding`

- `Magnitude`:
  - Round to digit at magnitude 10^m.
  - Contains m.

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::Magnitude(-2));
assert_eq!(f.format(123.456), "123,46");
assert_eq!(f.format(0.789), "790 m");
assert_eq!(f.format(42069), "42,06900 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::Magnitude(-1));
assert_eq!(f.format(123.456), "123,5");
assert_eq!(f.format(0.789), "800 m");
assert_eq!(f.format(42069), "42,0690 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::Magnitude(0));
assert_eq!(f.format(123.456), "123");
assert_eq!(f.format(0.789), "1");
assert_eq!(f.format(42069), "42,069 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::Magnitude(1));
assert_eq!(f.format(123.456), "120");
assert_eq!(f.format(0.789), "0");
assert_eq!(f.format(42069), "42,07 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::Magnitude(2));
assert_eq!(f.format(123.456), "100");
assert_eq!(f.format(0.789), "0");
assert_eq!(f.format(42069), "42,1 k");
```

- `SignificantDigits`:
  - Round to n significant numbers.
  - Contains n.

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::SignificantDigits(0));
assert_eq!(f.format(123.456), "0");
assert_eq!(f.format(0.789), "0");
assert_eq!(f.format(42069), "0");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::SignificantDigits(1));
assert_eq!(f.format(123.456), "100");
assert_eq!(f.format(0.789), "800 m");
assert_eq!(f.format(42069), "40 k");

let f: scaler::Formatter = scaler::Formatter::new()
```

```
    .set_rounding(scaler::Rounding::SignificantDigits(2));
assert_eq!(f.format(123.456), "120");
assert_eq!(f.format(0.789), "790 m");
assert_eq!(f.format(42069), "42 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::SignificantDigits(3));
assert_eq!(f.format(123.456), "123");
assert_eq!(f.format(0.789), "789 m");
assert_eq!(f.format(42069), "42,1 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::SignificantDigits(4));
assert_eq!(f.format(123.456), "123,5");
assert_eq!(f.format(0.789), "789,0 m");
assert_eq!(f.format(42069), "42,07 k");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_rounding(scaler::Rounding::SignificantDigits(5));
assert_eq!(f.format(123.456), "123,46");
assert_eq!(f.format(0.789), "789,00 m");
assert_eq!(f.format(42069), "42,069 k");
```

3.2. `Scaling`

- `Binary`:
  - Scales by factor $2^{10} = 1.024$.
  - If no prefix for that magnitude defined: Fallback to scientific notation.
  - Contains whether or not to put space between number and unit prefix.

```
‹› Rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::Binary(true));
assert_eq!(f.format(0.5), "1,000 * 2^(-1)");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(64), "64,00");
assert_eq!(f.format(128), "128,0");
assert_eq!(f.format(1023), "1.023");
assert_eq!(f.format(1024), "1,000 Ki");
assert_eq!(f.format(2_f64.powi(10)), "1,000 Ki");
assert_eq!(f.format(2_f64.powi(20)), "1,000 Mi");
assert_eq!(f.format(2_f64.powi(30)), "1,000 Gi");
assert_eq!(f.format(2_f64.powi(40)), "1,000 Ti");
assert_eq!(f.format(2_f64.powi(50)), "1,000 Pi");
assert_eq!(f.format(2_f64.powi(60)), "1,000 Ei");
assert_eq!(f.format(2_f64.powi(70)), "1,000 Zi");
assert_eq!(f.format(2_f64.powi(80)), "1,000 Yi");
assert_eq!(f.format(2_f64.powi(90)), "1,000 * 2^(90)");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::Binary(false));
assert_eq!(f.format(1024), "1,000Ki");
```

- `Decimal`:
  - Scales by factor $10^3 = 1.000$.
  - If no prefix for that magnitude defined: Fallback to scientific notation.
  - Contains whether or not to put space between number and unit prefix.

```
‹› Rust
```

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::Decimal(true));
assert_eq!(f.format(1e-31), "1,000 * 10^(-31)");
assert_eq!(f.format(1e-30), "1,000 q");
assert_eq!(f.format(1e-27), "1,000 r");
assert_eq!(f.format(1e-24), "1,000 y");
assert_eq!(f.format(1e-21), "1,000 z");
assert_eq!(f.format(1e-18), "1,000 a");
assert_eq!(f.format(1e-15), "1,000 f");
assert_eq!(f.format(1e-12), "1,000 p");
assert_eq!(f.format(1e-9), "1,000 n");
assert_eq!(f.format(1e-6), "1,000 µ");
assert_eq!(f.format(1e-3), "1,000 m");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(10), "10,00");
assert_eq!(f.format(100), "100,0");
assert_eq!(f.format(999), "999,0");
assert_eq!(f.format(1000), "1,000 k");
assert_eq!(f.format(1e3), "1,000 k");
assert_eq!(f.format(1e6), "1,000 M");
assert_eq!(f.format(1e9), "1,000 G");
assert_eq!(f.format(1e12), "1,000 T");
assert_eq!(f.format(1e15), "1,000 P");
assert_eq!(f.format(1e18), "1,000 E");
assert_eq!(f.format(1e21), "1,000 Z");
assert_eq!(f.format(1e24), "1,000 Y");
assert_eq!(f.format(1e27), "1,000 R");
assert_eq!(f.format(1e30), "1,000 Q");
assert_eq!(f.format(1e33), "1,000 * 10^(33)");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::Decimal(false));
assert_eq!(f.format(1000), "1,000k");
```

- `None`:
  - no scaling
  - no fallback to scientific notation

```rust
<> Rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::None);
assert_eq!(f.format(1e-10), "0,0000000001000");
assert_eq!(f.format(0.1), "0,1000");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(10), "10,00");
assert_eq!(f.format(100), "100,0");
assert_eq!(f.format(1000), "1.000");
assert_eq!(f.format(1e10), "10.000.000.000");
```

- `Scientific`:
  - always scientific notation

```rust
<> Rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::Scientific);
assert_eq!(f.format(0.1), "1,000 * 10^(-1)");
assert_eq!(f.format(1), "1,000 * 10^(0)");
assert_eq!(f.format(10), "1,000 * 10^(1)");
```

3.3. Separators

- `group_separator`

  - Separates groups every 3 digits before the decimal separator.
- `decimal_separator`
  - Separates the integer and fractional parts of a number.

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::None)
    .set_separators(".", ",");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(10), "10,00");
assert_eq!(f.format(100), "100,0");
assert_eq!(f.format(1000), "1.000");
assert_eq!(f.format(10000), "10.000");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::None)
    .set_separators("", ",");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(10), "10,00");
assert_eq!(f.format(100), "100,0");
assert_eq!(f.format(1000), "1000");
assert_eq!(f.format(10000), "10000");

let f: scaler::Formatter = scaler::Formatter::new()
    .set_scaling(scaler::Scaling::None)
    .set_separators(",", ".");
assert_eq!(f.format(1), "1.000");
assert_eq!(f.format(10), "10.00");
assert_eq!(f.format(100), "100.0");
assert_eq!(f.format(1000), "1,000");
assert_eq!(f.format(10000), "10,000");
```

3.4. `Sign`

- `Always`
  - Always show sign, even when number is positive.

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_sign(scaler::Sign::Always);
assert_eq!(f.format(std::f64::NEG_INFINITY), "−∞");
assert_eq!(f.format(-1), "-1,000");
assert_eq!(f.format(0), "+0,000");
assert_eq!(f.format(1), "+1,000");
assert_eq!(f.format(std::f64::INFINITY), "+∞");
```

- `OnlyMinus`
  - Only show sign when number is negative.

```rust
let f: scaler::Formatter = scaler::Formatter::new()
    .set_sign(scaler::Sign::OnlyMinus);
assert_eq!(f.format(std::f64::NEG_INFINITY), "−∞");
assert_eq!(f.format(-1), "-1,000");
assert_eq!(f.format(0), "0,000");
assert_eq!(f.format(1), "1,000");
assert_eq!(f.format(std::f64::INFINITY), "∞");
```