Table of Contents

1 double
1.1 Constants and Special Values
1.2 Basic Operations
1.3 Special Value Testing
1.4 Binary Representation

double

This package provides comprehensive support for double-precision floating-point arithmetic, including basic operations, trigonometric functions, exponential and logarithmic functions, as well as utility functions for handling special values
.

Constants and Special Values

The package provides several important constants and special floating-point valu
es:

```
1
2   test "special values" {
3
4     inspect(@double infinity, content="Infinity")
5     inspect(@double neg_infinity, content="-Infinity")
6     inspect(@double not_a_number, content="NaN")
7
8
9     inspect(@double max_value, content="1.7976931348623157e+308")
10    inspect(@double min_value, content="-1.7976931348623157e+308")
11    inspect(@double min_positive, content="2.2250738585072014e-308")
12  }
```

Basic Operations

Basic mathematical operations and rounding functions:

```
1
2   test "basic operations" {
3
4     inspect(@double abs(-3.14), content="3.14")
5
6
7     inspect(@double floor(3.7), content="3")
8     inspect(@double ceil(3.2), content="4")
9     inspect(@double round(3.5), content="4")
10    inspect(@double trunc(3.7), content="3")
11
12
13    inspect(2.0 pow(3), content="8")
14
15
16    inspect((-3.14) signum(), content="-1")
17    inspect(2.0 signum(), content="1")
18
19
20    inspect(@double from_int(42), content="42")
21  }
```

Special Value Testing

Functions for testing special floating-point values and comparing numbers:

```
1
2   test "special value testing" {
3
4     inspect(@double not_a_number is_nan(), content="true")
5     inspect(@double infinity is_inf(), content="true")
6     inspect(@double infinity is_pos_inf(), content="true")
7     inspect(@double neg_infinity is_neg_inf(), content="true")
8
9
10    let relative_tolerance = 1.e-9
11    inspect(@double is_close(0.1 + 0.2, 0.3, relative_tolerance~), content
12  }
```

Binary Representation

Functions for converting doubles to their binary representation:

```
1
2   test "binary representation" {
3     let num = 1.0
4
5
6
7     inspect(
8       num to_be_bytes(),
9       content=(
10        #|b"\x3f\xf0\x00\x00\x00\x00\x00\x00"
11      ),
12    )
13    inspect(
14      num to_le_bytes(),
15      content=(
16        #|b"\x00\x00\x00\x00\x00\x00\xf0\x3f"
17      ),
18    )
19  }
```

Note: Most methods can be called either as a method (d.to_be_bytes()) or as a package function (@double.to_be_bytes(d)).