1 Inequalities, Logical expressions

1.1 Boolean values

In mathematics it is common to test if an expression is true or false. For example, is the point $(1, 2)$ inside the disc $x^2 + y^2 \leq 1$? We would check this by substituting 1 for x and 2 for y, evaluating both sides of the inequality and then assessing if the relationship is true or false. In this case, we end up with a comparison of $5 \leq 1$, which we of course know is false.

`Julia` provides numeric comparisons that allow this notation to be exactly mirrored:

```
x, y = 1, 2
x^2 + y^2 <= 1
```

false

The response is `false`, as expected. `Julia` provides Boolean values `true` and `false` for such questions. The same process is followed as was described mathematically.

The set of numeric comparisons is nearly the same as the mathematical counterparts: `<`, `<=`, `==`, `>=`, `>`. The syntax for less than or equal can also be represented with the Unicode $\leq$ (generated by `\le[tab]`). Similarly, for greater than or equal, there is `\ge[tab]`.

> The use of `==` is necessary, as `=` is used for assignment.

The `!` operator takes a boolean value and negates it. It uses prefix notation:

```
!true
```

false

For convenience, `a != b` can be used in place of `!(a == b)`.

1.2 Algebra of inequalities

To illustrate, let's see that the algebra of expressions works as expected.

For example, if $a < b$ then for any c it is also true that $a + c < b + c$.

We can't "prove" this through examples, but we can investigate it by the choice of various values of a, b, and c. For example:

```
a,b,c = 1,2,3
a < b, a + c < b + c
```

```
(true, true)
```

Or in reverse:

```
a,b,c = 3,2,1
a < b, a + c < b + c
```

```
(false, false)
```

Trying other choices will show that the two answers are either both `false` or both `true`.

> Well, almost... When `Inf` or `NaN` are involved, this may not hold, for example `1 + Inf < 2 + Inf` is actually `false`.

So adding or subtracting any finite value from an inequality will preserve the inequality, just as it does for equations.

What about addition and multiplication?

Consider the case $a < b$ and $c > 0$. Then $ca < cb$. Here we investigate using 3 random values (which will be positive):

```
a,b,c = rand(3)   # 3 random numbers in (0,1)
a < b, c*a < c*b
```

```
(false, false)
```

Whenever these two commands are run, the two logical values should be identical, even though the specific values of `a`, `b`, and `c` will vary.

The restriction that $c > 0$ is needed. For example, if $c = -1$, then we have $a < b$ if and only if $-a > -b$. That is the inequality is "flipped."

```
a,b = rand(2)
a < b, -a > -b
```

```
(false, false)
```

Again, whenever this is run, the two logical values should be the same. The values a and $-a$ are the same distance from 0, but on opposite sides. Hence if $0 < a < b$, then b is farther from 0 than a, so $-b$ will be farther from 0 than $-a$, which in this case says $-b < -a$, as expected.

Finally, we have the case of division. The relation of x and $1/x$ (for $x > 0$) is that the farther x is from 0, the closer $1/x$ is to 0. So large values of x make small values of $1/x$. This leads to this fact for $a, b > 0$: $a < b$ if and only if $1/a > 1/b$.

We can check with random values again:

```
a,b = rand(2)
a < b, 1/a > 1/b
```

```
(true, true)
```

In summary we investigated numerically that the following hold:

- `a < b` if and only if `a + c < b + c` for all finite `a`, `b`, and `c`.
- `a < b` if and only if `c*a < c*b` for all finite `a` and `b`, and finite, positive `c`.
- `a < b` if and only if `-a > -b` for all finite `a` and `b`.
- `a < b` if and only if `1/a > 1/b` for all finite, positive `a` and `b`.

1.3 Some examples

We now show some inequalities highlighted on this Wikipedia page.

Numerically investigate the fact $e^x \geq 1 + x$ by showing it is true for three different values of x. We pick $x = -1$, 0, and 1:

```
x = -1; exp(x) >= 1 + x
x =  0; exp(x) >= 1 + x
x =  1; exp(x) >= 1 + x
```

true

Now, let's investigate that for any distinct real numbers, a and b that

$$\frac{e^b - e^a}{b - a} > e^{(a+b)/2}$$

For this, we use `rand(2)` to generate two random numbers in $(0, 1)$:

```
a, b = rand(2)
(exp(b) - exp(a)) / (b-a) > exp((a+b)/2)
```

true

This should evaluate to `true` for any random choice of `a` and `b` returned by `rand(2)`.

Finally, let's investigate the fact that the harmonic mean, $2/(1/a + 1/b)$ is less than or equal to the geometric mean, $\sqrt{ab}$, which is less than or equal to the quadratic mean, $\sqrt{a^2 + b^2}/\sqrt{2}$, using two randomly chosen values:

```
a, b = rand(2)
h = 2 / (1/a + 1/b)
g = (a * b) ^ (1 / 2)
q = sqrt((a^2 + b^2) / 2)
h <= g, g <= q
```

```
(true, true)
```

1.4 Chaining, combining expressions: absolute values

The absolute value notation can be defined through cases:

$$|x| = \begin{cases} x & x \geq 0 \\ -x & \text{otherwise.} \end{cases}$$

The interpretation of $|x|$, as the distance on the number line of x from 0, means that many relationships are naturally expressed in terms of absolute values. For example, a simple shift: $|x-c|$ is related to the distance x is from the number c. As common as they are, the concept can still be confusing when inequalities are involved.

For example, the expression $|x-5| < 7$ has solutions which are all values of x within 7 units of 5. This would be the values $-2 < x < 12$. If this isn't immediately intuited, then formally $|x-5| < 7$ is a compact representation of a chain of inequalities: $-7 < x - 5 < 7$. (Which is really two combined inequalities: $-7 < x - 5$ *and* $x - 5 < 7$.) We can "add" 5 to each side to get $-2 < x < 12$, using the fact that adding by a finite number does not change the inequality sign.

Julia's precedence for logical expressions, allows such statements to mirror the mathematical notation:

```
x = 18
abs(x - 5) < 7
```

false

This is to be expected, but we could also have written:

```
-7 < x - 5 < 7
```

false

Read aloud this would be "minus 7 is less than x minus 5 **and** x minus 5 is less than 7".

The "and" equations can be combined as above with a natural notation. However, an equation like $|x-5| > 7$ would emphasize an **or** and be "x minus 5 less than minus 7 **or** x minus 5 greater than 7". Expressing this requires some new notation.

The *boolean operators* `&` and `|` implement "and" and "or." Thus we could write $-7 < x-5 < 7$ as

```
(-7 < x - 5) & (x - 5 < 7)
```

false

and could write $|x-5| > 7$ as

```
(x - 5 < -7) | (x - 5 > 7)
```

true

(The first expression is false for $x = 18$ and the second expression true, so the "or"ed result is `true` and the "and" result if `false`.)

> The short circuit operators are `&&` and `||`. For simple Boolean values, they perform a related task, though have a more general usage.

Example One of DeMorgan's Laws states that "not (A and B)" is the same as "(not A) or (not B)". This is a kind of distributive law for "not", but note how the "and" changes to "or". We can verify this law systematically. For example, the following shows it true for 1 of the 4 possible cases for the pair `A`, `B` to take:

```
A,B = true, false  ## also true, true; false, true; and false, false
!(A & B) == !A | !B
```

true

1.5 Precedence

The question of when parentheses are needed and when they are not is answered by the precedence rules implemented. Earlier, we wrote

```
(x - 5 < -7) | (x - 5 > 7)
```

true

To represent $|x - 5| > 7$. Were the parentheses necessary? Let's just check.

```
x - 5 < -7 | x - 5 > 7
```

false

So yes, they were. The precedence rules perform `|` before `<` or `>`, so without the extra pair of parentheses, we would have

```
(x - 5 < ( (-7 | x) - 5)) > 7
```

false

which is not what is desired at all. (The value of `-7 | x` is `-5` - as `|` does something completely different when the two arguments are not boolean.)

A thorough understanding of the precedence rules can help eliminate unnecessary parentheses, but in most cases it is easier just to put them in.

1.6 Arithmetic with

For convenience, basic arithmetic can be performed with Boolean values, `false` becomes 0 and true 1. For example, both these expressions make sense:

```
true + true + false, false * 1000
```

```
(2, 0)
```

The first example shows a common means used to count the number of `true` values in a collection of Boolean values - just add them.

This can be cleverly exploited. For example, the following expression returns `x` when it is positive and 0 otherwise:

```
(x > 0) * x
```

18

There is a built in function, `max` that can be used for this: `max(0, x)`.

This expression returns `x` if it is between -10 and 10 and otherwise -10 or 10 depending on whether x is negative or positive.

```
(x < -10)*(-10) + (x >= -10)*(x < 10) * x + (x>=10)*10
```

10

The `clamp(x, a, b)` performs this task more generally, and is used as in `clamp(x, -10, 10)`.

1.7 Questions

⊛ Question
Is `e^pi` or `pi^e` greater?

1. `e^pi` is less than `pi^e`
2. `e^pi` is greater than `pi^e`
3. `e^pi` is equal to `pi^e`

⊛ Question
Is sin(1000) positive?

1. Yes
2. No

⊛ Question
Suppose you know $0 < a < b$. What can you say about the relationship between $-1/a$ and $-1/b$?

1. $$-1/a < -1/b$$

2. $$-1/a \geq -1/b$$

3. $$-1/a > -1/b$$

⊛ Suppose you know $a < 0 < b$, is it true that $1/a > 1/b$?

1. Yes, it is always true.
2. It is never true, as $1/a$ is negative and $1/b$ is positive
3. It can sometimes be true, though not always.

⊛ Question
The `airyai` function is a special function named after a British Astronomer who realized the function's value in his studies of the rainbow. The `SpecialFunctions` package must be loaded to include this function, which is done with the accompanying package `CalculusWithJulia`:

```
using CalculusWithJulia    # loads the `SpecialFunctions` package
airyai(0)
```

0 . 3 5 5 0 2 8 0 5 3 8 8 7 8 1 7 2

It is known that this function is always positive for $x > 0$, though not so for negative values of x. Which of these indicates the first negative value : `airyai(-1) <0`, `airyai(-2) < 0`, ..., or `airyai(-5) < 0`?

1. `airyai(-1) < 0`
2. `airyai(-2) < 0`
3. `airyai(-3) < 0`
4. `airyai(-4) < 0`
5. `airyai(-5) < 0`

⊛ Question
By trying three different values of $x > 0$ which of these could possibly be always true:

1. `x^x <= (1/e)^(1/e)`
2. `x^x == (1/e)^(1/e)`
3. `x^x >= (1/e)^(1/e)`

⊛ Question
Student logic says $(x+y)^p = x^p + y^p$. Of course, this isn't correct for all p and x. By trying a few points, which is true when $x, y > 0$ and $0 < p < 1$:

1. `(x+y)^p > x^p + y^p`
2. `(x+y)^p < x^p + y^p`
3. `(x+y)^p == x^p + y^p`

⊛ Question
According to Wikipedia, one of the following inequalities is always true for $a, b > 0$ (as proved by I. Ilani in JSTOR,AMM,Vol.97,No.1,1990). Which one?

1. `a^a + b^b >= a^b + b^a`
2. `a^a + b^b <= a^b + b^a`
3. `a^b + b^a <= 1`

⊛ Question
Is 3 in the set $|x - 2| < 1/2$?

1. Yes
2. No

⊛ Question
Which of the following is equivalent to $|x - a| > b$:

1. $ -b < x-a \text{ and } x - a < b$
2. $$-b < x - a < b$$
3. $ x - a < -b \text{ or } x - a > b$

⊛ Question
If $|x - \pi| < 1/10$ is $|\sin(x) - \sin(\pi)| < 1/10$?

Guess an answer based on a few runs of

```
x = pi + 0.2 * rand()
abs(x - pi) < 1/10, abs(sin(x) - sin(pi)) < 1/10
```

1. true
2. false

⊛ Question
Does `12` satisfy $|x-3|+|x-9|>12$?

1. Yes
2. No

⊛ Question
Which of these will show DeMorgan's law holds when both values are `false`:

1. `!(false & false) == false !& false`
2. `!(false & false) == !false & !false`
3. `!(false & false) == !false | !false`

⊛ Question
For floating point numbers there are two special values `Inf` and `NaN`. For which of these is the answer always `false`:

1. `NaN < 3.0` and `3.0 <= NaN`
2. `Inf < 3.0` and `3.0 <= Inf`

⊛ Question
The IEEE 754 standard is about floating point numbers, for which there are the special values `Inf`, `-Inf`, `NaN`, and, surprisingly, `-0.0` (as a floating point number and not `-0`, an integer). Here are 4 facts that seem reasonable:

- Positive zero is equal but not greater than negative zero.
- `Inf` is equal to itself and greater than everything else except `NaN`.
- `-Inf` is equal to itself and less then everything else except `NaN`.
- `NaN` is not equal to, not less than, and not greater than anything, including itself.

Do all four seem to be the case within `Julia`? Find your answer by trial and error.

1. Yes
2. No

⊛ Question
The `NaN` value is meant to signal an error in computation. `Julia` has value to indicate some data is missing or unavailable. This is `missing`. For `missing` values we have these computations:

```
(missing, true)
```

We see the value of `true | missing` is `true`. Why?

1. If `missing` were `true` or `false`, the answer would always be `true`
2. Since the second value is "`missing`", only the first is used. So `false | missing` would also be `false`

The value for `true & missing` is `missing` showing that there are possibly three values for `&` to return. Why would it make sense for this to be "`missing`"?

1. Since the second value is "`missing`" all such answers would be missing.
2. If `missing` were `true` the answer would be `true` and if it were `false` the answer would be `false`, so the answer is not known, hence also "`missing`"