# Выражения

## Оператор присваивания

Оператор присваивания в **Julia** имеет вид =.

Пример. Чтобы присвоить переменной a значение 1, нужно выполнить следующий код:

[ ]:
```
a = 1
```

Можно с помощью цепочки равенств присвоить значение 2 сразу двум переменным a и b:

[ ]:
```
a = b = 2
```

## Арифметические операторы

- +x – унарный плюс (операция тождественности);
- -x – унарный минус (изменение знака x на противоположный);
- x + y – сложение;
- x - y – вычитание;
- x * y – умножение;
- x / y – деление;
- x ÷ y – целочисленное деление (с округлением до целого числа);
- x \ y – деление с инверсией (эквивалентно y / x);
- x ^ y – возведение в степень (x в степени y);
- x % y – остаток от деления x на y;
- sqrt(x) – квадратный корень из x.

Приоритет действий стандартный: сначала выполняются унарные операции, затем возведение в степень, затем *, /, ÷, \, % и, наконец, сложение и вычитание.

Помещение числового множителя непосредственно перед переменной или скобками, например 2x или 2(x+y), рассматривается как умножение, причем с большим приоритетом, чем у других двоичных операций.

Операторы, выполняемые раньше других близлежащих операторов, отделяются меньшим расстоянием. Например, запись -x + 2 означает, что сначала выполняется инверсия x, а затем к результату прибавляется число 2.

Несколько простых примеров использования арифметических операторов:

[ ]:
```
543.6 + 987.6 + 321.2
```

[ ]:
```
1 - 2 + 3 - 4 + 5
```

[ ]:
```
3 * 2 / 12
```

[ ]:
```
100 ÷ 17
```

[ ]:
```
12 \ 3
```

[ ]:
```
pi^3
```

[ ]:
```
25 % 4
```

[ ]:
```
sqrt(1000)
```

**✏Задание 1**

Для переменных `a = 12.4` и `b = 71.3` выполните сложение, вычитание, умножение, деление a на b (оообычное, целочисленное и с инверсией) и возведение a в степень b.

[ ]:
```

```

**Решение**

[ ]:
```
a = 12.4
b = 71.3
a + b
```

[ ]:
```
a - b
```

[ ]:
```
a * b
```

[ ]:
```
a / b
```

[ ]:
```
a ÷ b
```

[ ]:
```
a \ b
```

[ ]:
```
a^b
```

Если необходимо изменить порядок действий, то используются круглые скобки `()`. В математическом выражении может быть сколько угодно вложенных друг в друга пар круглых скобок.

Пример. Вычислим значение выражения $(3,1+5,2\cdot 9,3)^{1+\frac{4}{7}}$. Для этого и основание, и показатель степени нужно взять в скобки:

[ ]:
```
(3.1+5.2*9.3)^(1+4/7)
```

**✏Задание 2**

Вычислите значение выражения $(5\pi^2 - \sqrt{3})^{\frac{7}{9}}$.

[ ]:
```
```

**Решение**

[ ]:
```
(5*pi^2-sqrt(3))^(7/9)
```

## Составные выражения

Иногда бывает удобно использовать выражение, в котором несколько подвыражений вычисляются по порядку и которое возвращает результат последнего подвыражения. Для этого в **Julia** есть две конструкции: блоки `begin` – `end` и цепочки через `;`. Значением обеих конструкций составных выражений является результат последнего подвыражения.

Вот пример блока `begin` – `end`:

[ ]:
```
z = begin
        x = 5
        y = 10
        x * y
    end
```

Так как эти выражения достаточно простые, их можно легко записать в одну строку, и здесь будет удобен синтаксис цепочки через `;`:

[ ]:
```
z = (x = 5; y = 10; x * y)
```

Такой синтаксис особенно полезен для сжатых однострочных определений функций.

Обычно блоки `begin` – `end` многострочные, а цепочки через `;` – однострочные, но это не строгое требование:

[ ]:
```
z = begin x = 5; y = 10; x * y end
```

[ ]:
```
z = (x = 5;
    y = 10;
    x * y)
```

## Логические операторы

Для типа `Bool` поддерживаются следующие логические операторы: * `!x` – отрицание (НЕ). Изменяет `true` на `false` и наоборот; * `x && y` – логическое умножение (конъюнкция, И); * `x || y` – логическое сложение (дизъюнкция, ИЛИ).

`Bool` педеставляет собой целочисленный тип, и для него также определяются все арифметические операторы и операторы сравнения.

**✏Задание 3**

Задайте две логические переменные `A = true` и `B = false`. Выполните для переменных `A` и `B` отрицание, логическое умножение и логическое сложение.

[ ]:

**Решение**

[ ]:
```
A = true
B = false
!A
```

[ ]:
```
!B
```

[ ]:
```
A && B
```

[ ]:
```
A || B
```

### Битовые операторы

Для всех примитивных целочисленных типов поддерживаются следующие битовые операторы:

- ~x – битовое НЕ;
- `x & y` – битовое И;
- `x | y` – битовое ИЛИ;
- `xor(x,y)` или `x ⊻ y` – битовое исключающее ИЛИ;
- ‘x ⊼ y’ – битовое И НЕ;
- `x ⊽ y` – битовое ИЛИ НЕ;
- `x >>> y` – логический сдвиг вправо;
- `x >> y` – арифметический сдвиг вправо;
- `x << y` – логический/арифметический сдвиг влево.

Примеры использования битовых операторов:

[ ]:
```
~253
```

[ ]:
```
253 & 342
```

[ ]:
```
253 | 342
```

[ ]:
```
xor(253,342)
```

[ ]:
```
253 ⊻ 342
```

[ ]:
```
253 ⊼ 342
```

[ ]:
```
253 ⊽ 342
```

**✏Задание 4**

Над числами 78 и 61 выполните следующие битовые операции: битовое НЕ, битовое И, битовое ИЛИ, битовое исключающее ИЛИ, битовое И НЕ, битовое ИЛИ НЕ.

[ ]:

**Решение**

[ ]: ~78

[ ]: ~61

[ ]: 78 & 61

[ ]: 78 | 61

[ ]: xor(78,61)

[ ]: 78 ⊻ 61

[ ]: 78 ⊼ 61

Все двоичные арифметические и битовые операторы имеют **версию с присваиванием**, которая передает результат операнду, стоящему слева. Для использования версии двоичного оператора с присваиванием необходимо сразу после оператора указать знак =. Оператор с присваиванием переопределяет значение переменной в левой части. Например, запись `x += 3` эквивалентна записи `x = x + 3`.

Пример. Сначала присвоим переменной x значение 1, затем прибавим к нему 3 и результат сохраним в той же переменной x.

[ ]:
```
x = 1
x += 3
```

Все версии двоичных арифметических и битовых операторов с присваиванием:

`+=` `-=` `*=` `/=` `\=` `÷=` `%=` `^=` `&=` `|=` `⊻=` `>>>=` `>>=` `<<=`

## Векторные операторы с точкой

Для каждой двоичной операции (+, -, *, /, ^ и т.д.), имеется соответствующая операция с точкой, т.е. `.+`, `.-`, `.*`, `./`, `.^` и т.д. В этом случае операции выполняются поэлементно для каждого значения массива. Например, операция `[1, 2, 3] ^ 3` не определена, т.к. не существует стандартной математической операции возведения в куб для неквадратного массива. Однако определена операция

[ ]:
```
[1, 2, 3] .^ 3
```

Эта операция поэлементно вычисляет векторный результат `[1^3, 2^3, 3^3]`.

Операции с точкой позволяют сочетать массивы со скалярными значениями, сочетать друг с другом массивы одинакового размера и даже массивы разных размеров (например, сочетать горизонтальные и вертикальные векторы, создавая матрицу).

При сочетании точечных операторов с числовыми выражениями может возникать неоднозначность. Например, что означает `1.+x`? Это может быть `1. + x` или `1 .+ x`. Поэтому такой синтаксис не разрешен; в таких случаях нужно ставить вокруг оператора пробелы.

### ✏Задание 5

Задайте массив `x = [5 3 8 7 2]`. Вычислите массив y по формуле $y = (x^2+3x+1)^5$.

[ ]:
```

```

**Подсказка** Перед операторами сложения, умножения и возведения в степень нужно ставить точку: `.+`, `.*`, `.^`.

Решение

[ ]:
```
x = [5 3 8 7 2]
y = (x .^ 2 .+ 3 .* x .+ 1) .^ 5
```

### Операторы сравнения

Для всех примитивных числовых типов определены следующие операции сравнения:

- == – равенство;
- !=, ≠ – неравенство;
- < – меньше;
- <=, ≤ – меньше или равно;
- > – больше;
- >=, ≥ – больше или равно.

Результат выполнения этих операций имет логический тип (`true` или `false`).

Несколько простых примеров использования операторов сравнения:

[ ]:
```
5 == 5
```

[ ]:
```
1 == 0
```

[ ]:
```
1 != 2
```

[ ]:
```
1<2
```

[ ]:
```
3>=3
```

[ ]:
```
10 < -10
```

[ ]:
```
-1 <= -2
```

### ✏Задание 6

Вычислите значения логических выражений:

а) $\frac{12}{35} > \frac{13}{37}$

б) $\frac{324}{85} < 3,8$

в) $624 \cdot 379 = 236496$

[ ]:
```

```

**Решение**

[ ]:
```
12/35 > 13/37
```

[ ]:
```
324/85 < 3.8
```

[ ]:
```
624*379 == 236496
```

### Цепочки сравнений

В отличие от большинства языков, в **Julia** возможны произвольные цепочки сравнений.

Например:

[ ]:
```
1 < 2 < 3
```

Результат будет равен `true`, если результат всех сравнений в этой цепочке – истина, и `false` – в противном случае.

## Тест для получения сертификата

Пройти тест по теме **Выражения**.