Массивы

Массив – это коллекция объектов, хранящихся в многомерной таблице.

Одномерные массивы

В зависимости от способа создания **одномерный массив** в **Julia** может иметь тип `Vector` (вектор) или `Matrix` (матрица).

Вектор задается перечислением его элементов через запятую или точку с запятой в квадратных скобках. Например:

[ ]:
```
a = [1, 2, 3, 4, 5]
```

или

[ ]:
```
a = [1; 2; 3; 4; 5]
```

Элементами массивов могут быть не только целые числа, как в приведенных примерах, но и элементы любого другого типа (вещественные, комплексные, рациональные, логические, символьные, строковые и т.д.). Они также могут быть результатами вычисления каких-либо математических или логических выражений.

✏Задание 1

Создайте вектор b, содержащий числа 25, 76, 43 и 81 ииименно в таком порядке.

[ ]:
```

```

Решение Числа можно разделять запятыми:

[ ]:
```
b = [25, 76, 43, 81]
```

или точкой с запятой:

[ ]:
```
b = [25; 76; 43; 81]
```

Матрица-строка создается с помощью перечисления ее элементов через пробел в квадратных скобках. Например:

[ ]:
```
a = [1 2 3 4 5]
```

✏Задание 2

Создайте матрицу-строку b, содержащую результаты вычисления следующий выражений: $\sqrt{2}$, 3π, $9/11$ и 1.001^{1000} именно в таком порядке.

[ ]:

Решение

[ ]:
```
b = [sqrt(2) 3*pi 9/11 1.001^1000]
```

Двумерные массивы

Двумерные массивы имеют несколько строк и столбцов. Ввод элементов осуществляется по строкам, элементы одной строки отделяются друг от друга пробелами, а строки отделяются точкой с запятой. Все двумерные массивы в **Julia** имеют тип `Matrix`.

Например, зададим двумерный массив, содержащий 3 строки и 5 столбцов:

[ ]:
```
a = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]
```

Задание арифметических последовательностей

Оператор :

Часто бывает нужно создать вектор или матрицу, содержащие упорядоченные последовательности чисел. Упорядоченная последовательность создается с помощью оператора `:` для определения диапазона, заданного своими границами: начало`:`конец. Шаг между числами по умолчанию равен 1.

Например, выражение `2:5` задает диапазон целых чисел от 2 до 5 с шагом 1, т.е. 2, 3, 4, 5.

Однако, если мы введем команду `x = 2:5` или `x = [2:5]`, то у нас получится не вектор, а объект типа **диапазон** (**range**). Он будет хранить не сами элементы диапазона, а их **функцию-генератор**:

[ ]:
```
x = [2:5]
```

Если после диапазона добавить точку с запятой (`;`), то мы получим объект типа `Vector`:

[ ]:
```
x = [2:5;]
```

✏Задание 3

Задайте вектор, заполненный целыми числами от 10 до 20 с шагом 1, с помощью оператора диапазона `:`.

[ ]:

Решение

[ ]:
```
x = [10:20;]
```

Оператор диапазона позволяет задавать числа с произвольным шагом с помощью такой конструкции:

`начало:шаг:конец.`

Пример. С помощью оператора диапазона зададим вектор целых чисел от 10 до 20 с шагом 2:

[ ]:
```
x = [10:2:20;]
```

✏Задание 4

С помощью оператора диапазона задайте вектор, заполненный числами от 1 до 2 с шагом 0,1.

[ ]:

Решение

[ ]:
```
x = [1:0.1:2;]
```

Функция `collect`

Еще один способ преобразовать диапазон в вектор – использование функции `collect`.

Например, диапазон целых чисел от 1 до 10 с шагом 1 может быть преобразован в объект типа `Vector` следующим образом:

[ ]:
```
collect(1:10)
```

Функция `LinRange`

Если известно количество элементов, из которого должен состоять вектор, то задать диапазон равномерно распределенных чисел можно при помощи функции `LinRange`:

`LinRange(начало, конец, количество_элементов)`

Здесь для разделения аргументов команды используются запятые (`,`).

Пример. Зададим диапазон от 0 до 10, сотстоящий из 5 чисел:

[ ]:
```
x = LinRange(0, 10, 5)
```

В результате получается объект типа `LinRange`. Чтобы преобразовать его в тип `Vector`, воспользуемся знакомой нам функцией `collect`:

[ ]:
```
collect(x)
```

✏Задание 5

Создайте диапазон из 10 чисел, равномерно распределенных на отрезке от 5 до 32. Преобразуйте результат в тип `Vector`.

[ ]:
```

```

Решение

[ ]:
```
x = collect(LinRange(5, 32, 10))
```

Задавая векторы в **Julia**, мы всегда получаем векторы-стобцы. Если нам необходимо получить вектор-строку, то можно воспользоваться операцией трнспонирования (`'`).

Пример:

[ ]:
```
x = [1:5;]
x'
```

Заметим, что вектор стал не матрицей, а объектом типа `adjoint`. Обычно это не создает проблем при вычислениях.

Операции с массивами

Чтобы к каждому элементу массива прибавить одно и то же число, нужно использовать оператор сложения с точкой (`.+`). Например:

[ ]:
```
x = [1 2 3]
y = x .+ 4
```

Аналогично выполняется вычитание одного и того же числа из каждого элемента массива. Например:

[ ]:
```
y = x .- 5
```

Если нужно умножить или разделить все элементы массива на скаляр, то можно воспользоваться обычным синтаксисом:

[ ]:
```
y = x*2
```

[ ]:
```
y = x / 10
```

Чтобы сложить два массива одинаковой длины, синтаксис для поэлементного сложения `.+` тоже сработает, но проще сложить векторы обычным образом, с помощью оператора +. То же самое относится и к вычитанию массивов.

[ ]:
```
z = x + y
```

[ ]:
```
w = x - y
```

✏Задание 6

Создайте два вектора `x = [11 12 13 14]` и `y = [15 16 17 18]`. Вычислите: * сумму вектора x и числа 10; * произведение вектора y и числа 3; * сумму векторов x и y.

[ ]:
```

```

Решение

[ ]:
```
x = [11 12 13 14]
y = [15 16 17 18]
println(x .+ 2)
println(y*3)
println(x + y)
```

В **Julia** большинство функций можно применить к массивам, используя поэлементную нотацию (с точкой после имени функции).

Например, квадратный корень из каждого элемента массива x вычисляется так:

[ ]:
```
x = [1 2 3]
sqrt.(x)
```

✏Задание 7

Вычислите значение синуса каждого элемента массива `x = [1 2 3]`, воспользовавшись функцией `sin.` (с точкой).

[ ]:
```

```

Решение

[ ]:
```
x  = [1 2 3]
y = sin.(x)
```

Оператор * осуществляет **матричное умножение**. Для этого число столбцов в первой матрице должно быть равно числу строк во второй. В противном случае матричное умножение будет невозможным.

Пример. Умножим матрицу $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ на вектор-столбец $B = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$.

[ ]:
```
A = [1 2; 3 4]
B = [5; 6]
A * B
```

С помщью оператора `.*` можно выполнить поэлементное умножение массивов одинакового размера. Например:

[ ]:
```
A = [1 2]
B = [3 4]
A .* B
```

Индексация элементов массивов

Получить значение какого-либо элемента одномерного массива можно, обращаясь к его индексу в квадратных скобках. Нумерация элементов в **Julia** начинается с единицы.

Пример. Зададим вектор x и вызовем его третий и пятый элементы:

[ ]:
```
x = [238 435 546 659 412 3456 562 534 433 234]
a = x[3]
```

[ ]:
```
b = x[5]
```

Чтобы получить первый элемент массива, можно использовать как индекс 1, так и ключевое слово `begin`.

Пример. Вызовем первый элемент массива x:

[ ]:
```
c = x[begin]
```

Чтобы получить последний элемент массива, можно использовать ключевое слово end.

Пример. Вызовем последний и предпоследний элементы массива x:

[ ]:
```
d = x[end]
```

[ ]:
```
h = x[end-1]
```

Для вызова элемента двумерного массива `A` используются два индекса через запятую: `A[i,j]`. Первый индекс `i` означает номер строки, второй индекс `j` – номер столбца, на пересечении которых находится этот элемент.

Пример. Зададим матрицу A и присвоим переменной a значение элемента, находящегося во 2-й строке и 3-м столбце матрицы A:

[ ]:
```
A = [1 2 3; 4 5 6; 7 8 9]
a = A[2,3]
```

Можно изменить значения выбранных элементов массива, присвоив им новые значения.

Пример. Присвоим элементу A[1,1] значение 100 и выведем на экран обновленную матрицу A:

[ ]:
```
A[1,1] = 100
display(A)
```

✏Задание 8

Задайте матрицу A размером 4×4 и заполните ее последовательностью натуральных чисел от 1 до 16. * Присвойте переменной x значение элемента матрицы A, находящегося на пересечении 3-й строки и 1-го столбца; * Присвойте переменной y значение элемента матрицы A, находящегося на пересечении последней строки и предпоследнего слолбца (воспользуйтесь индексом end).

[ ]:
```

```

Решение

[ ]:
```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
x = A[3,1]
y = A[end,end-1]
println(x)
println(y)
```

Можно индексировать элементы матриц при помощи одномерного индекса. Тогда **Julia** использует такой порядок индексации: счетчик элементов будет последовательно отсчитывать их вдоль столбцов, начиная с первго, пока не дойдет до элемента с заданным индексом. Например, следующий код вернет значение 6.

[ ]:
```
A = [5 6;
     7 8]
A[3]
```

Оператор двоеточие (:) можно использовать в качестве индекса массива. Тогда он будет обозначать, что нужно вернуть все элементы по некоторой размерности массива. Например, следующий код создаст вектор-столбец, в котором содержатся все элементы из первого столбца матрицы A.

```
[ ]: A = [1 2 3; 4 5 6; 7 8 9]
```

```
[ ]: x = A[:, 1]
```

Оператор : можно использовать для задания диапазона индексов. Вот, например, код, который возвращает матрицу, состоящую из первой, второй и третьей строк некоторой матрицы B.

```
[ ]: B = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
```

```
[ ]: x = B[1:3,:]
```

✏Задание 9

- Создайте вектор x, содержащий 2-ю строку матрицы B, заданной в предыдущем примере.
- Создайте вектор y, содержащий со 2-го по 3-й столбцы той же матрицы B.

```
[ ]:
```

Решение

```
[ ]: x = B[2,:]
```

```
[ ]: y = B[:, 2:3]
```

Индексацию не обязательно производить при помощи непрерывных рядов. Например, следующий код извлекает первый, третий и шестой элементы вектора x. При этом в качестве индекса передается список `[1 3 6]`.

```
[ ]: x = [100:110;]
```

```
[ ]: x[[1 3 6]]
```

Функции для работы с массивами

Общие функции

- `eltype(A)` – тип элементов в массиве A;
- `length(A)` – количество элементов в массиве A;
- `ndims(A)` – количество измерений массива A;
- `size(A)` – кортеж, содержащий размеры массива A по всем измерениям;
- `size(A,n)` – кортеж, содержащий размер массива A по измерению n.

Пример. Создадим двумерный массив и определим тип и количество элементов в нем, количество измерений и его размеры по всем измерениям.

[ ]:
```
A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]
eltype(A)
```

[ ]:
```
length(A)
```

[ ]:
```
ndims(A)
```

[ ]:
```
size(A)
```

В **Julia** доступно множество функций для создания и инициализации массивов. В приведенном ниже списке вызовы с аргументом `dims...` могут принимать либо один кортеж с размерами измерений, либо набор размеров измерений, передаваемый в виде переменного числа аргументов. Большинство таких функций также принимают первый аргумент T, определяющий тип элементов массива. Если аргумент T не указан, по умолчанию используется тип `Float64`.

- `zeros(T, dims...)` – массив размером `dims` со всеми элементами, равными нулю;
- `ones(T, dims...)` – массив размером `dims` со всеми элементами, равными 1;
- `trues(dims...)` – массив размером `dims` со всеми элементами, равными `true`;
- `falses(dims...)` – массив размером `dims` со всеми элементами, равными `false`;
- `copy(A)` – копирует массив A;
- `reinterpret(T, A)` – массив с теми же двоичными данными, что и в A, но с типом элементов T;
- `rand(T, dims...)` – массив размером `dims` со случайными, независимо, одинаково и равномерно распределенными значениями. Для типов с плавающей запятой T значения находятся в интервале $[0, 1)$;
- `randn(T, dims...)` – массив размером `dims` со случайными, независимо и одинаково распределенными значениями, имеющими стандартное нормальное распределение (с математическим ожиданием, равным нулю, и со средним квадратическим отклонением, равным 1);
- `Matrix{T}(I, m, n)` – матрица, у которой на главной диагонали находятся единицы, а остальные элементы – нули (матрицы тождественности) размером $m \times n$. Для `I` требуется подключение библиотеки `LinerarAlgebra`;
- `range(start, stop, n)` – диапазон из `n` элементов, распределенных с равным шагом от `start` до `stop`;
- `fill(x, dims...)` – заполняет массив размером `dims` значениями `x`.

Примеры. Создадим двумерный массив размером 4×5, заполненный нулями. Ниже показаны три способа записи соответствующей команды.

[ ]:
```
A = zeros(Int8, 4, 5)
```

[ ]:
```
A = zeros(Int8, (4, 5))
```

[ ]:
```
A = zeros(4, 5)
```

Создадим массив размером 3×3 с равномерно распределенными случайными числами в интервале $[0, 1)$:

[ ]:
```
A = rand(3, 3)
```

Создадим единичную матрицу размером 7×7:

[ ]:
```
using LinearAlgebra
A = Matrix{Int8}(I, 7, 7)
```

Заполним массив размером 3×4 числами 5:

[ ]:
```
fill(5, (3,4))
```

✏Задание 10

С помощью функции ones создайте массив размером 5×6 со всеми элементами, равными 1.

[ ]:
```

```

Решение

[ ]:
```
A = ones(5, 6)
```

✏Задание 11

С помощью функции randn создайте массив размером 10×4 со случайными, независимо и одинаково распределенными значениями, имеющими стандартное нормальное распределение.

[ ]:
```

```

Решение

[ ]:
```
A = randn(10, 4)
```

Изменение размеров массива

- `reshape(A, dims...)` – массив с теми же данными, что и в A, но с другими измерениями;
- `transpose(A)` – транспонирование массива A;
- `cat(A..., dims)` – выполняет конкатенацию (объединение) входных массивов в измерениях, указанных в dims. Если dims=1, то это vcat; если dims=2, то hcat;
- `vcat(A...)` – выполняет конкатенацию массивов или чисел по вертикали. Эквивалентна `cat(A...; dims=1)`;

- `hcat(A...)` – выполняет конкатенацию массивов или чисел по горизонтали. Эквивалентна `cat(A...; dims=2)`;

Пример. Зададим одномерный массив размером из 16 элементов и с помощью функции reshape изменим его размер так, чтобы он стал двумерным и содержал 2 строки и 8 столбцов.

[ ]:
```
x = [1:16;]
```

[ ]:
```
reshape(x, 2, 8)
```

✏Задание 12

С помощью функции reshape измените размер массива x, заданного в предыдущем примере, так, чтобы он содержал 4 строки и 4 столбца.

[ ]:
```

```

Решение

[ ]:
```
reshape(x, 4, 4)
```

✏Задание 13

С помощью функций hcat и vcat выполните горизонтальную и вертикальную конкатенацию двумерных массивов $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ и $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$.

[ ]:
```

```

Решение

[ ]:
```
A = [1 2; 3 4]
B = [5 6; 7 8]
C = hcat(A, B)
```

[ ]:
```
D = vcat(A, B)
```

Функции для работы с элементами массива

- `in(x, A)` или `x in A` – проверяет, находится ли элемент x в массиве A.

[ ]:
```
A = [3 4 1 5 2]
in(3, A)
```

[ ]:
```
7 in A
```

- `maximum(A)` – находит максимальный элемент массива A.

[ ]:
```
maximum(A)
```

- `minimum(A)` – находит минимальный элемент массива A.

[ ]:
```
minimum(A)
```

- `sum(A)` – суммирование элементов массива A.

[ ]:
```
sum(A)
```

- `sum(A, dims=n)` – суммирование элементов двумерного массива A по столбцам (если `dims=1`) или по строкам (если `dims=2`).

[ ]:
```
A = [1 2 3; 4 5 6; 7 8 9]
sum(A, dims=1)
```

[ ]:
```
sum(A, dims=2)
```

- Элементы массива можно заполнить с помощью функциональной зависимости по следующей схеме:

`A = [выражение for индекс in начало:шаг:конец]`

Переменная индекс пробегает все значения в диапазоне от начало до конец с заданным шагом (если шаг равен 1, то его можно не писать). Значения элементов массива определяются с помощью выражения, которое, в частности, может зависеть от переменной индекс.

Например, зададим одномерный массив с десятью элементами, заполненный квадратами натуральных чисел от 1 до 10.

[ ]:
```
A = [i^2 for i in 1:10]
```

Таким же способом можно задавать двумерные массивы. Например, заполним двумерный масссив произведениями чисел x*y, если x изменятется от 1 до 5, а y – от 6 до 9.

[ ]:
```
A = [x*y for x = 1:5, y = 6:9]
```

- `sort(A, dims=n)` – производит сортировку элементов массива A по указанной размерности в порядке возрастания.
- `sort(A, dims=n, rev=true)` – производит сортировку элементов массива A по указанной размерности в порядке убывания.

Для одномерного веткора-строки нужно указывать `dims=2`. Ниже приведены примеры сортировки элементов одномерного массива:

[ ]:
```
x = [7 4 1 9 6 0 5 8 3 2]
sort(x, dims=2)
```

[ ]:
```
sort(x, dims=2, rev=true)
```

Аналогичная функция `sort!` (с восклицательным знаком) записывает результат в векторх.

✏Задание 14

Задайте массив `x = [39 21 62 81 54 47]`. Затем: * найдите его макисимальный и минимальный элементы, * найдите сумму его элементов, * отсортируйте его элементы в порядке возрастания и в порядке убывания.

[ ]:

Решение

[ ]:
```
x = [39 21 62 81 54 47]
maximum(x)
```

[ ]:
```
minimum(x)
```

[ ]:
```
sum(x)
```

[ ]:
```
sort(x, dims=2)
```

[ ]:
```
sort(x, dims=2, rev=true)
```

Бинарные маски

Индексировать элементы массивов можно с помощью **бинарных масок** – логических массивов типа BitVector или BitMatrix. В таком случае **Julia** возвращает только те элементы массива, которые соответствуют элементам логического массива, равным 1 (истина).

Пример. Зададим вектор x и с помощью бинарной маски выберем из него только те элементы, значения которых превышают 6.

[ ]:
```
x = [1 5 7 8 2 4 10 3 6]
y = x[x .> 6]
```

Перед знаком логического оператора (>, >=, <, <=, ==, !=) обязательно нужно ставить точку, т.к. логическая операция производится над каждым элементом массива x.

✏Задание 15

Создайте переменную z, содержащую элементы вектора x, значения которых меньше или равны 5.

[ ]:

Решение

[ ]:
```
z = x[x .<= 5]
```

Логическую индексацию по значениям одного вектора можно использовать и для индексации другого вектора.

В следующем примере в переменную с помещаются значения вектора a, соответствующие тем позициям в векторе b, при которых значения элементов вектора b больше 6.

[ ]:
```
a = [1 2 3 4 5]
b = [4 5 6 7 8]
c = a[b .> 6]
```

✏Задание 16

Поместите в переменную с все элементы вектора b, соответствующие тем позициям в векторе, при которых значения элементов вектора a меньше 4.

[ ]:
```

```

Решение

[ ]:
```
c = b[a .< 4]
```

Вы можете использовать логические массивы, чтобы индексировать другие массивы, например для работы с отдельными их элементами.

Например, если вам нужно заменить значения всех элементов массива x, значения которых больше 9, на значение 0, синтаксис будет следующим:

[ ]:
```
x = [1 5 7 8 2 4 10 3 6]
x[x .> 9] .= 0
println(x)
```

Кроме операторов сравнения, вы можете использовать логические операторы AND (&) и OR (|), чтобы совместить несколько операций сравнения.

Например, чтобы получить все элементы, значения которых меньше 8 **и одновременно** больше 3, используйте оператор & (AND):

[ ]:
```
y = x[(x .< 8) .& (x .> 3)]
```

Чтобы получить все элементы, значения которых больше 6 **или** меньше 2, используйте оператор | (OR):

[ ]:
```
y = x[(x .> 6) .| (x .< 2)]
```

✏Задание 17

Создайте переменную y, содержащую элементы вектора x = [17 25 36 42 53 68 71 84], значения которых больше 20, но меньше 70.

[ ]:
```
```

Решение

[ ]:
```
x = [17 25 36 42 53 68 71 84]
y = x[(x .> 20) .& (x .< 70)]
```

Тест для получения сертификата

Пройти тест по теме **Массивы**.