ВЫСОКОУРОВНЕВЫЙ ВЫСОКОПРОИЗВОДИТЕЛЬНЫЙ СВОБОДНЫЙ

ВАДИМ НИКИТИН

julia

ЯЗЫК ПРОГРАММИРОВАНИЯ

БЫСТРЫЙ СТАРТ

2024 1.10+

Вадим Никитин

Julia. Язык программирования. Быстрый старт

«Автор»
2023

Никитин В.

Julia. Язык программирования. Быстрый старт / В. Никитин — «Автор», 2023

Начните писать программы на Julia в первый же день.Необходимы только минимальные знания в использовании компьютеров и программировании. Вы знаете что такое файл или каталог? Вам известны алгоритмы? Этого достаточно.Ничего лишнего. В этом справочнике раскрыты самые важные и часто затрагиваемые темы в краткой и ясной форме, а для желающих погрузиться в тему глубже представлены ссылки на необходимые источники информации.Подробные инструкции не упускают ни единого шага, а наглядные примеры полностью отображают в себе весь процесс работы. Вы всегда будете знать, что должно получиться в результате. Все примеры были протестированы на актуальной версии языка программирования Julia, доступной на момент написания данного справочника.Удобный дизайн оглавления позволяет быстро находить нужные примеры функции и команд. Нет нужды держать все в голове или постоянно лезть в интернет.Теперь второе издание с добавлением нового и обновлением старого.

Содержание

Начало работы 5
Рабочая среда Julia 6
Основной режим 7
Справочная система 8
Менеджер пакетов 9
Системная оболочка 10
Поиск по истории 11
Программы Julia 12
Дополнительные инструменты 13
Visual Studio Code 14
Jupyter 15
Pluto.jl 16
Пакеты в Julia 17
Использование диспетчера пакетов 18
Использование пакетов 19
Дополнительная информация 20
Переменные 21
Типы данных 24
Целые числа 26
Беззнаковые целые числа 28
Поведение при переполнении 30
Числа с плавающей точкой 31
Ноль с плавающей точкой 33
Специальные значения 34
Тип NaN 36
Машинный эпсилон 38
Базовые операторы Julia 39
Логические операторы 41
Побитовые операторы 42
Операторы обновления 43
Векторизированные “точечные” операторы 45
Конец ознакомительного фрагмента. 46

Вадим Никитин
Julia. Язык программирования. Быстрый старт

Начало работы

Для того, чтобы приступить к работе, вам необходимо загрузить и установить дистрибутив Julia для вашей операционной системы (Windows, macOS, Linux, FreeBSD), следуя инструкциям на сайте https://julialang.org/downloads/. Некоторые дистрибутивы Linux включают Julia и Juliaup в состав своих пакетов, но убедитесь, что вы устанавливаете актуальную версию.

Это руководство написано на основе версии Julia 1.10.0, которая является текущей выпущенной версией Julia. Концепции, описанные в книге, носят общий характер и применимы к более поздним версиям языка. Однако, возможно, что некоторые выходные данные более поздних версий Julia могут не соответствовать результатам и примерам, представленным в этой книге.

Самый простой способ изучать и экспериментировать с Julia – это запустить интерактивный сеанс рабочей среды, дважды щелкнув на исполняемом файле Julia или запустить julia из командной строки:

```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.0 (2023-12-25)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia>
```

Чтобы закончить интерактивный сеанс, используйте команду exit() или комбинацию клавиш CTRL + D.

Рабочая среда Julia

Рабочая среда Julia – REPL (от англ. read-eval-print loop – «цикл „чтение – вычисление – вывод“»), оболочка с полнофункциональной интерактивной командной строкой, встроенная в исполняемый файл julia. С помощью этой оболочки мы взаимодействуем с JIT-компилятором (англ. Just-in-Time, компиляция «точно в нужное время») для тестирования и запуска нашего кода, помимо этого доступна история команд с возможностью поиска, автодополнение с помощью табуляции, множество полезных привязок клавиш, а также специальные режимы справки и оболочки. REPL имеет пять режимов работы.

Основной режим

Это режим работы по умолчанию, каждая новая строка изначально начинается с приглашения julia>. Именно здесь вы можете вводить выражения Julia от простых до многострочных конструкций. Нажатие клавиши Return или Enter после ввода выражения запускает выполнение и вывод результата. Например:

```
julia> 2 + 2
4
julia> 5 * (5 – 1)
20
julia> 5/2
2.5
```

Или сообщения об ошибке если что то пошло не так:

```
julia> 5/"A"
ERROR: MethodError: no method matching /(::Int64, ::String)
```

Справочная система

Julia имеет встроенную справочную систему, которая извлекает информацию об использовании большинства функций непосредственно из исходного кода. Это справедливо и для большинства сторонних пакетов.

Для перехода в справочную систему Julia наберите ? (знак вопроса) в начале строки основного режима. Приглашение командной строки примет вид:

help?>

Julia попытается найти и отобразить справку или документацию для всего, что было введено в режиме справки. Если вы не помните точное название функции, Julia вернет список похожих функций. Хотя фактическое возвращаемое содержимое может различаться, вы можете ожидать увидеть следующую информацию для каждой запрашиваемой функции:

- Написание
- Однострочное описание
- Список аргументов
- Подсказки к аналогичным или связанным функциям
- Один или несколько примеров использования
- Список методов (для функций, которые имеют несколько реализаций)

Возврат в режим по умолчанию производится нажатием комбинации клавиш CTRL-C или клавишей BACKSPACE в начале строки.

Дополнительно в дистрибутив Julia входит локальная копия официального сайта документации https://docs.julialang.org/en/v1/ расположенная:

[JULIA_INSTALL_FOLDER]/share/doc/julia/html/en (где JULIA_INSTALL_FOLDER – каталог, куда установлен Julia)

Что позволяет использовать сайт документации в системах изолированных от интернета.

Менеджер пакетов

После установки Julia вы получите компилятор, который преобразует написанный вами код на Julia в версию, которую может выполнить ваш компьютер, а так же стандартную библиотеку (иногда называемую Base), которая содержит базовую функциональность, встроенную в среду. Сюда входят такие вещи, как массивы и списки, числа и строки, некоторые основы линейной алгебры и статистики и т.д. Но в большинстве случаев, скорее всего, вам может потребоваться расширить функционал, загрузив и запустив внешние библиотеки, которые в Julia называются пакетами. Для работы с ними используется Pkg – встроенный менеджер пакетов Julia, который выполняет такие операции как: установка, обновление и удаление пакетов. Вход в него осуществляется нажатием клавиши] в основном режиме. Приглашение командной строки примет вид:

```
(@v1.10) pkg>
```

Возврат в режим по умолчанию – нажатием комбинации клавиш CTRL-C или клавишей BACKSPACE в начале строки.

Системная оболочка

Режим позволяет использовать командную оболочку операционной системы для выполнения системных команд. Для перехода наберите ; (точка с запятой) в начале строки основного режима. Приглашение командной строки примет вид:

shell>

Возврат в режим по умолчанию – нажатием комбинации клавиш CTRL-C или клавишей BACKSPACE в начале строки.

Для пользователей Windows режим оболочки Julia не предоставляет команд оболочки windows напрямую, дополнительно необходимо задать командную оболочку PowerShell или cmd.exe.

● PowerShell:

```
shell> powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
PS C:\Users\julia>
```

● cmd.exe:

```
shell> cmd
Microsoft Windows [version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Users\julia>
```

Поиск по истории

Во всех вышеперечисленных режимах выполненные строки сохраняются в файл истории, по которому можно производить поиск. Чтобы инициировать инкрементный поиск по предыдущей истории, нажмите CTRL-R. Подсказка изменится на (reverse-i-search)`': . По мере ввода поисковый запрос будет появляться между символами `'. Самый последний результат, соответствующий запросу, будет динамически обновляться справа от двоеточия по мере набора текста. Чтобы найти более старый результат по тому же запросу, просто введите CTRL-R еще раз.

Так же как CTRL-R – поиск по истории назад, CTRL-S – поиск по истории вперед, с подсказкой (forward-i-search)`': . Эти две функции можно использовать в сочетании друг с другом для перехода к предыдущему или следующему результату поиска соответственно.

Все выполненные команды в REPL записываются в ~/.julia/logs/repl_history.jl вместе с меткой времени, когда они были выполнены, и текущим режимом REPL, в котором вы находились. Режим поиска запрашивает этот файл журнала, чтобы найти команды, которые вы выполняли ранее.

Программы Julia

В то время как интерактивные выражения – это быстрый способ попробовать что-то в REPL и просмотреть результаты, реальные приложения требуют выполнения больших фрагментов кода. В Julia вы вводите весь программный код в обычный текстовой файл с расширением .jl. Запустите любой текстовый редактор на ваш вкус и введите следующий код в файл:

```
msg="Hello, World!"
println(msg)
```

Сохраните файл как example.jl в папке, где развернута Julia (только лишь для удобства, чтобы не указывать полный путь к файлу в команде). Запустите julia. Затем в командной строке julia вызовите команду, указанную ниже, и вы должны увидеть вывод:

```
julia> include("example.jl")
Hello, World!
```

Другой способ запустить программу Julia – запустить ее из командной строки терминала операционной системы. Откройте терминал в папке где развернута Julia (опять же лишь для удобства, чтобы не указывать полный путь к файлу в команде), наберите julia example.jl и нажмите Enter и вы снова должны увидеть вывод:

```
[vadim@void-linux julia-1.10.0]$ julia example.jl
Hello, World!
[vadim@void-linux julia-1.10.0]$
```

Программа: в первой строке мы создали переменную msg и присвоили ей текстовое значение, во второй строке функция println() выводит ее значение в поток вывода по умолчанию.

Вы можете определить глобальный (для всех пользователей компьютера) и локальный (для одного пользователя) файл программы Julia, который будет выполняться при любом запуске Julia, где могут быть определены функции или переменные, которые всегда должны быть доступны. Расположение этих двух файлов следующее:

- Глобальный файл Julia: [JULIA_INSTALL_FOLDER]\etc\julia\startup.jl (где JULIA_INSTALL_FOLDER – каталог, где установлена Julia)

- Локальный файл Julia: [USER_HOME_FOLDER]\.julia\config\startup.jl (где USER_HOME_FOLDER – домашний каталог локального пользователя, например %HOMEPATH% в Windows и ~ в Linux)

Обратите внимание, что локальный каталог конфигурации может не существовать. В этом случае вы можете просто создать отсутствующие каталоги и файл своими руками.

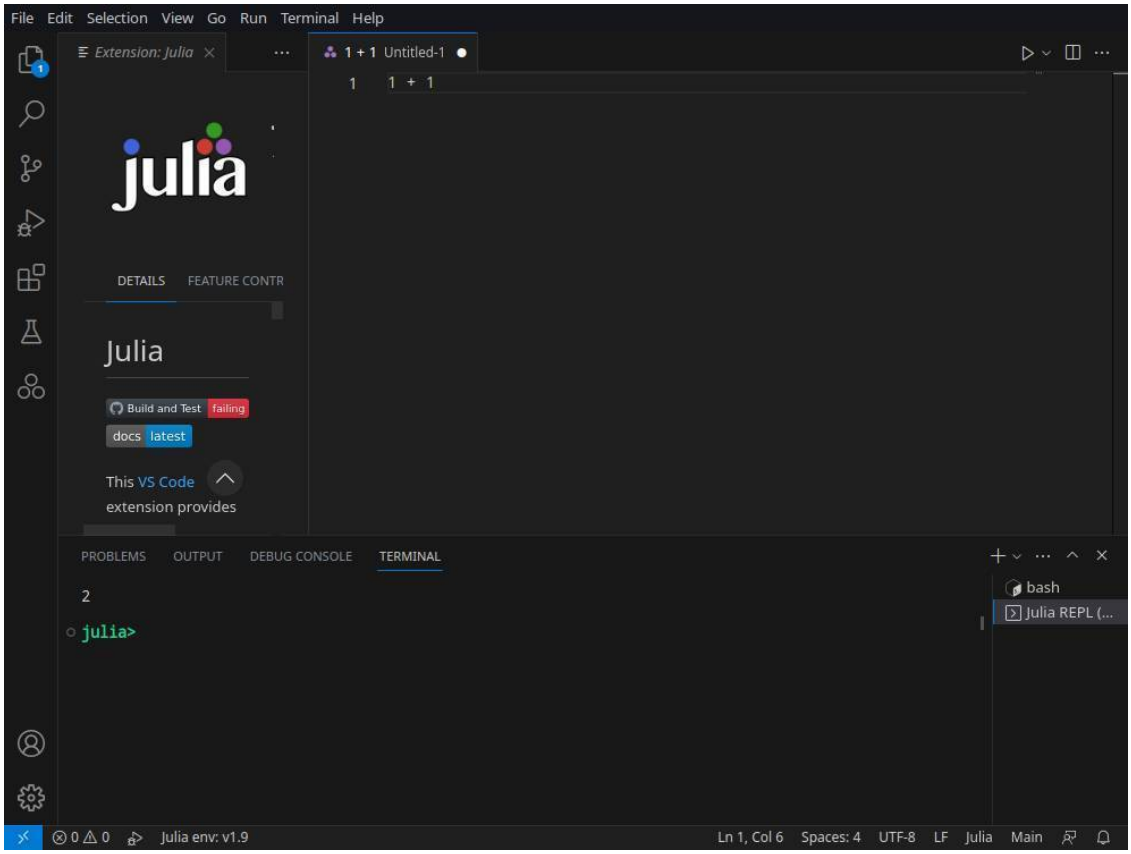
Дополнительные инструменты

Работа с кодом в обычном текстовом редакторе это конечно вопрос вкуса. Существует множество различных инструментов, созданных с целью облегчить и ускорить процесс работы в области программирования. Ниже приведен пример самых популярных в сообществе инструментов, которые регулярно обновляются, являются бесплатными, а также поддерживаются самими разработчиками языка программирования Julia.

Visual Studio Code

Visual Studio Code (сокращенно VS Code) – это мощный текстовой редактор исходного кода. Его отличительные особенности ниже:

- Подсветка синтаксиса и технология умного автодополнения.
- Отладка кода прямо в редакторе.
- Инструменты для работы с Git.
- Очень широкие возможности кастомизации.



Начните работу здесь: https://www.julia-vscode.org

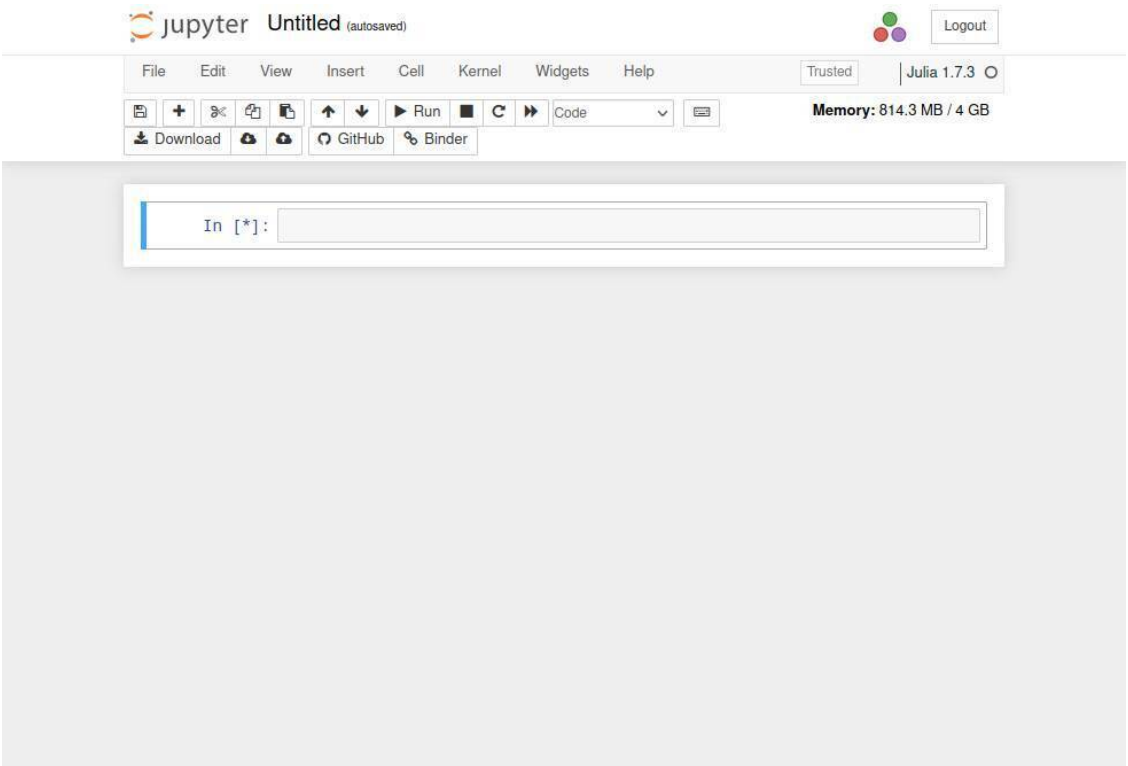
Jupyter

JupyterLab и Jupyter Notebook – это веб приложения позволяющие работать с Julia и другими языками программирования через браузер. По сути, такие приложения представляют собой онлайн блокноты с дополнительными функциями. Некоторые из особенностей Jupyter:

● Блокнотами можно делиться с другими с помощью электронной почты, Dropbox, GitHub и Jupyter Notebook Viewer.

● Ваш код может создавать богатый интерактивный вывод: HTML, изображения, видео, LaTeX и пользовательские типы MIME.

● Централизованное развертывание



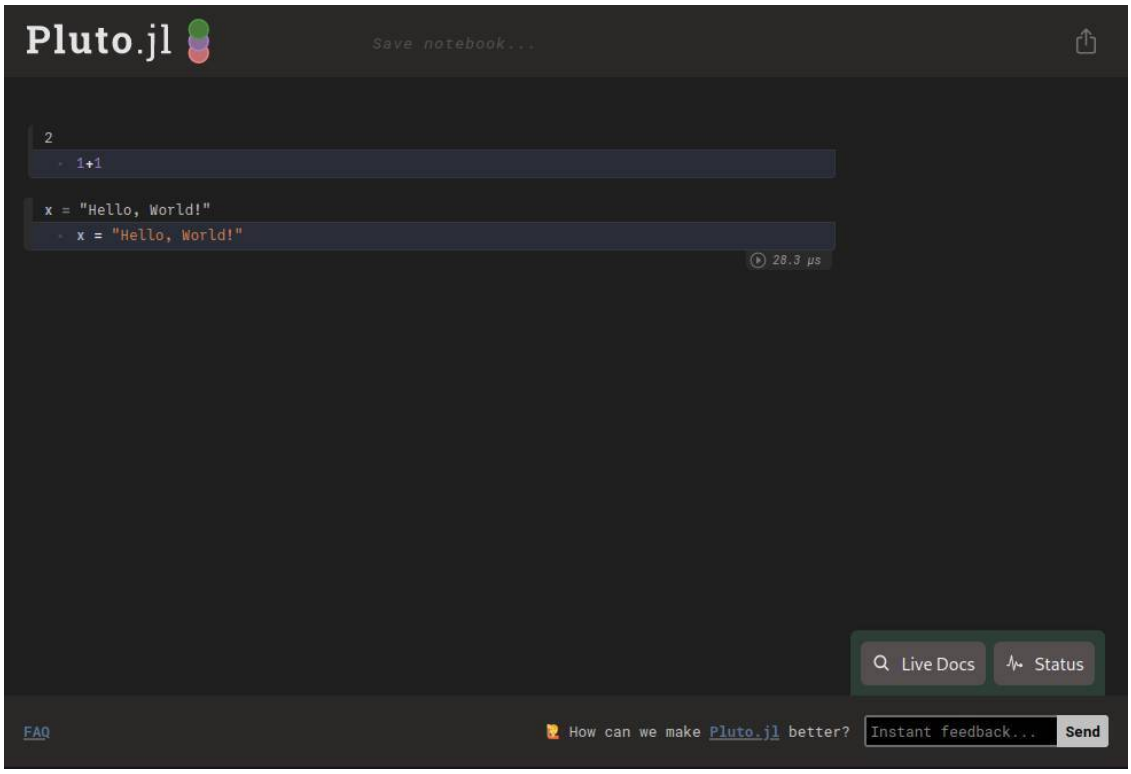
Pluto.jl

Pluto – это другой блокнот, созданный специально для работы с языком программирования Julia. Блокнот Pluto состоит из небольших блоков кода Julia (ячеек), и вместе они образуют реактивный блокнот. Когда вы меняете переменную, Pluto автоматически перезапускает ячейки, которые на нее ссылаются. Ячейки можно размещать в произвольном порядке – интеллектуальный синтаксический анализ определяет зависимости между ними и позаботится о выполнении. Некоторые отличительные черты Pluto:

● Реактивность – при изменении функции или переменной Pluto автоматически обновляет все затронутые ячейки.

● Pluto написан на Julia и прост в установке.

● Прост в использовании.



Пакеты в Julia

Julia имеет модульную конструкцию – компактное ядро, функциональность которого расширяется внешними «пакетами». Дистрибутив Julia поставляется с небольшим набором этих пакетов, называемым «стандартной библиотекой», и мощным менеджером пакетов, который может загружать пакеты, предварительно компилировать, обновлять и разрешать зависимости, и все это с помощью нескольких простых команд.

В то время как зарегистрированные пакеты можно установить, просто используя их имя, для незарегистрированных пакетов необходимо указать их исходное местоположение. На момент написания этого текста было опубликовано более 9000 зарегистрированных пакетов. Знание того, как работают пакеты, крайне необходимо для эффективной работы в Julia, и именно поэтому управление пакетами представлено в начале руководства.

Использование диспетчера пакетов

Есть два способа получить доступ к функциям управления пакетами, в интерактивном режиме или через API из кода программы Julia:

● 1.Интерактивный способ – ввести] в консоли REPL, чтобы войти в «особый» режим pkg. Приглашение изменится с julia> на (vX.Y) pkg>, где vX.Y – версия активной среды Julia. Затем вы можете запустить любые команды диспетчера пакетов или вернуться в обычный режим интерпретатора с помощью комбинации клавиш CTRL-C или клавиши BACKSPACE в начале строки.

● 2.Способ API заключается в том, чтобы импортировать модуль Pkg в код программы (using Pkg), а затем выполнить команду Pkg.<команда менеджера пакетов>(<аргументы команды>). Очевидно, ничто не мешает вам использовать подход API и в интерактивном сеансе, но в специальном пакетном режиме есть автозавершение и другие полезные функции, которые делают его более удобным в использовании.

Обратите внимание, что два интерфейса не на 100 % совместимы, а интерфейс API несколько более строгий.

Некоторые из полезных команд диспетчера пакетов:

● status: Извлекает список (имя и версию) локально установленных пакетов.

● update: Обновляет локальный индекс пакетов и все локальные пакеты до последней версии.

● add <имя пакета>: Автоматически загружает и устанавливает заданный пакет. Для нескольких пакетов используйте add <имя пакета 1> <имя пакета 2>.

● add <имя пакета>#master, add <имя пакета>#branchName или add <имя пакета>#vX.Y.Z: Извлекает главную ветвь данного пакета, определенную ветвь или определенный выпуск соответственно.

● free <имя пакета>: Возвращает пакет к последнему выпуску.

● rm <имя пакета>: Удаляет пакет и все зависимые от него пакеты, которые были автоматически установлены только для него.

● add https://github.com/<имя репозитория>/<имя пакета>.jl: Извлекает незарегистрированный пакет по URL-адресу (здесь это GitHub).

Использование пакетов

Чтобы получить доступ к функциональным возможностям установленного пакета, вам необходимо использовать команду using или import. Разница между ними заключается в следующем:

● Использование пакета позволяет получить прямой доступ к функциям пакета. Просто используйте команду using <имя пакета> в консоли REPL или поместите в начало файла скрипта.

● Импорт пакета делает то же самое, но помогает поддерживать чистоту пространства имен, так как затем вам нужно обращаться к функциям пакета, используя их полные имена <имя пакета>.<имя функции>. Вы можете использовать псевдонимы или выбрать импорт только подмножества функций (к которым вы затем сможете получить прямой доступ).

Например, чтобы получить доступ к функции now() из пакета Dates (идет в комплекте дистрибутива), вы можете сделать следующее:

● Получите прямой доступ к функциям пакета с помощью using <имя пакета> :

```
julia> using Dates
julia> now()
2023-05-13T20:23:03.187
```

● Получите доступ к функциям пакета, используя их полные имена, с помощью import <имя пакета>:

```
julia> import Dates
julia> Dates.now()
2023-05-13T20:43:04.801
```

● Получите прямой доступ к функциям пакета с помощью import <имя пакета>:<имя функции> :

```
julia> import Dates:now
julia> now()
2023-05-13T20:46:53.542
```

Наконец, вы также можете получить доступ к функциям любого исходного файла Julia, используя эту строку:

```
include("<путь к файлу><имя файла>.jl")
```

Когда эта строка выполняется, включенный файл полностью запускается (не только анализируется), и любой определенный там символ становится доступным в области видимости (область кода, в которой видна переменная) относительно того места, где было вызвано включение.

Дополнительная информация

Для получения справки по всем командам пакетного менеджера введите ? или help в приглашении:

(@v1.10) pkg> help

Для получения справки по отдельной команде с примерами введите ? <имя команды> в приглашении, например:

(@v1.10) pkg> ?add

Полное руководство по менеджеру пакетов, доступно на официальном сайте https://pkgdocs.julialang.org

Актуальные и рекомендуемые сервисы для навигации по экосистеме пакетов на официальном сайте https://julialang.org/packages/

Переменные

Переменная в Julia – это имя, привязанное к значению. Она используется, когда вы хотите сохранить значение (полученное, например, после математических вычислений) для последующего использования. Например:

● Присвоить значение 100 переменной x

```
julia> x = 100
100
```

● Выполнение математических операций со значением x

```
julia> x * 5
500
```

● Переназначить значение x

```
julia> x = 10 + 10
20
```

● Можно присваивать и значения других типов, например, строки текста

```
julia> x = "Hello World!"
"Hello World!"
```

Присвоение "имя = значение" привязывает переменную имени к значению, вычисленному в правой части, и все присваивание рассматривается Julia как выражение, равное значению правой части. Это означает, что присваивания можно объединять (одно и то же значение присваивается нескольким переменным переменная1 = переменная2 = значение) или использовать в других выражениях, а также то, почему их результат отображается в REPL как значение правой части. Например, здесь значение 4 из b = 2+2 используется в другой арифметической операции и присваивании:

```
julia> a = (b = 2 + 2) * 5
20

julia> a
20

julia> b
4
```

При знакомстве с переменными в Julia у новых пользователей часто возникает путаница между присвоением имени и изменением значения. Если вы выполнили a = 2, а затем a = 3, то вы изменили имя a, чтобы оно ссылалось на новое значение 3. Вы не изменили число 2, поэтому 2+2 по-прежнему дает 4, а не 6! Это различие становится более очевидным при работе с мутабельными типами данных, такими как массивы, содержимое которых может быть изменено:

```
julia> a=[1,2,3]
3-element Vector{Int64}:
1
2
3

julia> b=a
3-element Vector{Int64}:
1
2
3
```

Здесь строка b = a не создает копию массива a, а просто связывает имя b с тем же массивом: a и b "указывают" на один массив [1,2,3] в памяти.

Изменим значение первого элемента массива:

```
julia> a[1] = 42
42
```

Присваивание a[i] = value изменяет содержимое массива, измененный массив будет виден через имена a и b:

```
julia> a
3-element Vector{Int64}:
42
2
3

julia> b
3-element Vector{Int64}:
42
2
3
```

Пусть a теперь является именем другого объекта:

```
julia> a= 3.14159
3.14159
```

Установка a = 3.14159 не изменяет массив, а просто привязывает a к другому объекту, массив по-прежнему доступен через b:

```
julia> b
3-element Vector{Int64}:
42
2
3
```

Имена переменных в Julia могут быть любой длины,а также могут содержать в себе почти все символы Unicode, но не могут начинаться с цифры. В именах можно использовать прописные и строчные буквы, символ подчеркивания ('_') также может использоваться в имени переменной в любом месте. Имена переменных чувствительны к регистру.

Единственными явно запрещенными именами переменных являются имена встроенных ключевых слов: baremodule, begin, break, catch, const, continue, do, else, elseif, end, export, false, finally, for, function, global, if, import, let, local, macro, module, quote, return, struct, true, try, using, while.

Примеры допустимых и недопустимых имен:

```
julia> x1 = 100
100

julia> 1x=100
ERROR: syntax: "1" is not a valid function argument name around REPL[2]:1

julia> ●="Точка"
"Точка"

julia> text@ = "Строка текста"
ERROR: syntax: extra token "@" after end of expression
```

Типы данных

По умолчанию Julia автоматически определяет какой тип данных использовать для значения переменной, но в некоторых случая, во избежании ошибок, следует указать тип данных для значений вручную.

Ниже приведен пример такой ошибки. Здесь функция typeof() возвращающая тип аргумента, а sqrt() – корень квадратный из аргумента:

```
julia> x=-2.0
–2.0

julia> typeof(x)
Float64

julia> sqrt(x)
ERROR: DomainError with -2.0:
sqrt will only return a complex result if called with a complex argument. Try sqrt(Complex(x)).
```

Происходит следующее: Julia автоматически определяет тип значения переменной, как Float64, исходя из того, что в большинстве случаев используются действительные числа, а не комплексные, что и вызывало ошибку. Теперь тот же пример используя комплексную форму записи:

```
julia> x=-2.0+0im
–2.0 + 0.0im

julia> typeof(x)
ComplexF64 (alias for Complex{Float64})

julia> sqrt(x)
0.0 + 1.4142135623730951im
```

В этом случае Julia определяет тип значения исходя из формы записи как Complex{Float64}, по сути формой записи мы задали тип значения переменной.

Типы есть только у значений. У переменных типов нет. Переменные – это просто имена, связанные со значениями, хотя для простоты можно говорить "тип переменной", как сокращение от "тип значения, на которое ссылается переменная".

Julia изначально предоставляет довольно полный и иерархически организованный набор предопределенных типов, особенно числовых. Это либо скаляры, такие как: целые числа (Int), числа с плавающей запятой (Float) и символы (Char). Либо контейнероподобные структуры, способные хранить другие объекты, такие как: многомерные массивы (Array), словари (Dict), наборы (Set) и т. д. По стилистическим соглашениям названия типов начинаются с заглавной буквы, например Int64 или Bool. Иногда в фигурных скобках за именем типа следуют другие параметры, например типы содержащихся элементов или количество измерений. Эти параметры встречаются у всех контейнероподобных структур и некоторых неконтейнерных. Например тип Array{Int64,2} будет использоваться для двумерного массива целых 64-битных чисел со знаком. В терминологии Julia такие типы называются параметрическими.

Оператор :: можно использовать для присоединения аннотаций типов к выражениям и переменным в программах, например:

```
julia> (2+2)::Int
4

julia> (2+2)::AbstractFloat
ERROR: TypeError: in typeassert, expected AbstractFloat, got a value of type Int64

julia> (2.0+2.0)::AbstractFloat
4.0
```

При добавлении к выражению, вычисляющему значение, оператор :: читается как "является экземпляром". Его можно использовать в любом месте, чтобы утверждать, что значение выражения слева является экземпляром типа справа. Если тип справа конкретный, то значение слева должно иметь этот тип в качестве своей реализации. Если тип абстрактный, то достаточно, чтобы значение было реализовано конкретным типом, который является подтипом абстрактного типа. Если утверждение о типе не истинно, выбрасывается исключение, в противном случае возвращается левое значение.

Пример используемый выше, теперь нет нужды использовать комплексную форму записи:

```
julia> x::Complex{Float64}=-2
–2

julia> typeof(x)
ComplexF64 (alias for Complex{Float64})

julia> sqrt(x)
0.0 + 1.4142135623730951im
```

Когда оператор :: добавляется к переменной в левой части присваивания, он означает немного другое: он объявляет переменную всегда имеющей указанный тип, как объявление типа в статически типизированном языке, таком как C. Каждое значение, присвоенное переменной, будет преобразовано к объявленному типу с помощью функции convert(), если это возможно.

Целые числа

Типы целых чисел в Julia:

Тип	Знак	Количество битов	Наименьшее значение	Наибольшее значение
Int8	✓	8	-2^7	2^7-1
UInt8		8	0	2^8-1
Int16	✓	16	-2^15	2^15-1
UInt16		16	0	2^16-1
Int32	✓	32	-2^31	2^31-1
UInt32		32	0	2^32-1
Int64	✓	64	-2^63	2^63-1
UInt64		64	0	2^64-1
Int128	✓	128	-2^127	2^127-1
UInt128		128	0	2^128-1
Bool	N/A	8	false(0)	true(1)

Целые числа вводятся и выводятся стандартным образом:

```
julia> 10
10

julia> 0123456789
123456789
```

Тип по умолчанию для целых чисел зависит от разрядности системы (64-бита):

```
julia> typeof(10)
Int64

julia> typeof(0123456789)
Int64
```

Julia также определяет типы Int и UInt, которые являются псевдонимами для системных знаковых и беззнаковых типов целых чисел соответственно:

```
julia> Int
Int64

julia> UInt
UInt64
```

Большие целые числа, которые не могут быть представлены с использованием 64 бит, но могут быть представлены в 128 битах, всегда создают 128-битные целые числа, независимо от типа системы:

```
julia> typeof(10000000000000000000)
Int128
```

Беззнаковые целые числа

Беззнаковые целые числа вводятся и выводятся с использованием префикса 0x и цифр от 0 до 9, а также латинских букв от a до f, используемых для обозначения шестнадцатеричных чисел, использование заглавных букв A-F также допустимо. Размер беззнакового значения определяется количеством используемых шестнадцатеричных цифр:

```
julia> x = 0x1
0x01

julia> typeof(x)
UInt8

julia> x = 0x123
0x0123

julia> typeof(x)
UInt16

julia> x = 0x1234567
0x01234567

julia> typeof(x)
UInt32

julia> x = 0x123456789abcdef
0x0123456789abcdef

julia> typeof(x)
UInt64

julia> x = 0x111122223333444455556666777788888
0x111122223333444455556666777788888

julia> typeof(x)
UInt128
```

Значения, слишком большие для типов Int128, UInt128, при вводе получат специальный тип BigInt:

```
julia> typeof(100000000000000000000000000000000000000)
Int128

julia> typeof(1000000000000000000000000000000000000000)
BigInt

julia> typeof(0xffffffffffffffffffffffffffffffff)
UInt128
```

```
julia> typeof(0xffffffffffffffffffffffffffffffff)
BigInt
```

Это не беззнаковый тип, но это единственный встроенный тип, достаточно большой для представления таких больших целых значений.

Поведение при переполнении

В Julia превышение максимального представляемого значения данного типа приводит к циклическому поведению. Пример (функции typemax(), typemin(), возвращают максимальное и минимальное значения для заданного типа, == оператор равенства):

```
julia> x = typemax(Int64)
9223372036854775807

julia> x+1
–9223372036854775808

julia> x + 1 == typemin(Int64)
true

julia> x = typemax(UInt64)
0xffffffffffffffff

julia> x+1
0x0000000000000000

julia> x + 1 == typemin(UInt64)
true
```

В тех случаях, когда переполнение возможно, рекомендуется производить проверку на цикличное поведение. В противном случае используйте тип BigInt арифметики произвольной точности. Ниже приведен пример поведения при переполнении и как его можно решить с помощью BigInt():

```
julia> 10^19
–8446744073709551616

julia> BigInt(10)^19
10000000000000000000
```

Числа с плавающей точкой

Типы чисел с плавающей точкой в Julia:

Тип	Точность	Количество битов
Float16	Половинная	16
Float32	Одинарная	32
Float64	Двойная	64

Числа с плавающей точкой вводятся и выводятся стандартным образом:

```
julia> 1.0
1.0

julia> 1.
1.0

julia> 0.5
0.5

julia> .5
0.5

julia> -1.23
–1.23
```

При необходимости можно использовать E-нотацию:

```
julia> 1e10
1.0e10

julia> 2.5e-4
0.00025
```

Все результаты из примеров выше имеют тип Float64 (тип по умолчанию). Если вы хотите ввести значение с типом Float32, то необходимо использовать f вместо e следующим образом:

```
julia> x = 0.5f0
0.5f0

julia> typeof(x)
Float32

julia> 2.5f-4
```

```
0.00025f0
```

Значение с типом Float16:

```
julia> Float16(4.)
Float16(4.0)

julia> 2*Float16(4.)
Float16(8.0)
```

Ноль с плавающей точкой

Числа с плавающей точкой имеют два нуля – положительный нуль и отрицательный нуль. Они равны друг другу, но имеют разные двоичные представления, что можно увидеть с помощью функции bitstring(), которая дает буквальное битовое представление примитивного типа:

```
julia> 0.0 == -0.0
true

julia> bitstring(0.0)
"0000000000000000000000000000000000000000000000000000000000000000"

julia> bitstring(-0.0)
"1000000000000000000000000000000000000000000000000000000000000000"
```

Когда точности или размерности Float64 недостаточно, можно использовать специальный тип BigFloat:

```
julia> 2.0^100/4
3.1691265005705735e29

julia> BigFloat(2.0)^100/4
3.169126500570573503741758013 44e+29
```

BigFloat знаковый тип арифметики произвольной точности, не назначаемый автоматически при вводе, а требующий явного объявления для использования.

Функции минимального и максимального значений для типов также применимы:

```
julia> (typemin(Float16),typemax(Float16))
(-Inf16, Inf16)

julia> (typemin(Float32),typemax(Float32))
(-Inf32, Inf32)

julia> (typemin(Float64),typemax(Float64))
(-Inf, Inf)
```

Результатом будут специальные значения – отрицательная и положительная бесконечности. Значения чисел превышающих числовой диапазон типа также будут заменены на специальные значения:

```
julia> 4.2^1000
Inf

julia> -4.2^1000
–Inf
```

Специальные значения

Существует три определенных стандартных значения с плавающей точкой, которые не соответствуют ни одной точке на линии вещественных чисел:

Float16	Float32	Float64	Имя	Описание
Inf16	Inf32	Inf	Положительная бесконечность	Значение больше чем все конечные значения с плавающей запятой
-Inf16	-Inf32	-Inf	Отрицательная бесконечность	Значение меньше, чем все конечные значения с плавающей запятой
NaN16	NaN32	NaN	Не число	Значение не == любому значению с плавающей запятой (включая себя)

По стандарту IEEE 754, эти значения с плавающей точкой являются результатами определенных арифметических операций:

```
julia> 1/0
Inf

julia> -5/0
–Inf

julia> 0.000001/0
Inf

julia> 0/0
NaN

julia> 1/Inf
0.0

julia> 1/-Inf
–0.0

julia> -1/Inf
–0.0

julia> -1/-Inf
0.0

julia> 500 + Inf
Inf
```

```
julia> 500 – Inf
–Inf

julia> Inf + Inf
Inf

julia> -Inf -Inf
–Inf

julia> Inf – Inf
NaN

julia> Inf * Inf
Inf

julia> Inf*-Inf
–Inf

julia> -Inf * -Inf
Inf

julia> Inf / Inf
NaN

julia> Inf /-Inf
NaN

julia> -Inf /Inf
NaN

julia> -Inf /-Inf
NaN

julia> 0 * Inf
NaN

julia> 0 *-Inf
NaN
```

Тип NaN

NaN не равно, не меньше и не больше чего-либо, включая самого себя:

```
julia> NaN == NaN
false

julia> NaN != NaN
true

julia> NaN < NaN
false

julia> NaN > NaN
false
```

Это может вызвать проблемы, например при работе с массивами:

```
julia> [1 NaN] == [1 NaN]
false
```

Функции Julia для работы со специальными значениями:

Функция	Описание
isequal(x, y)	Проверяет, идентичны ли x и y. Возвращает значение Bool.
isfinite(x)	Проверяет, является ли x конечным. Возвращает значение Bool.
isinf(x)	Проверяет, является ли x бесконечным. Возвращает значение Bool.
isnan(x)	Проверяет, является ли x NaN. Возвращает значение Bool.

Функция isequal() считает NaNs равными друг другу:

```
julia> isequal(NaN, NaN)
true

julia> isequal([1 NaN], [1 NaN])
true

julia> isequal(NaN, NaN32)
true
```

Функцию isequal() можно также использовать для различения знаковых нулей:

```
julia> -0.0 == 0.0
true
```

```
julia> isequal(-0.0, 0.0)
false
```

Машинный эпсилон

Большинство реальных чисел не могут быть точно представлены числами с плавающей точкой, поэтому для многих целей важно знать расстояние между двумя соседними представляемыми числами с плавающей точкой, которое часто называют машинным эпсилоном.

Функция eps() в Julia дает расстояние между 1.0 и следующим большим значением с плавающей точкой, при использовании в качестве аргумента типа числа с плавающей точкой:

```
julia> eps(Float16)
Float16(0.000977)

julia> eps(Float32)
1.1920929f-7

julia> eps(Float64)
2.220446049250313e-16

julia> eps(BigFloat)
1.72723371101888892507727037256007991422320007288725627700474069403371836063248 5e
```

Функция eps также может принимать в качестве аргумента значение с плавающей точкой, и выдавать абсолютную разницу между этим значением и следующим представимым значением с плавающей точкой. Другими словами, eps(x) выдает значение того же типа, что и x, такое, что x + eps(x) является следующим представимым значением с плавающей точкой, большим, чем x. Тип значения при этом также учитывается:

```
julia> eps(1.0)
2.220446049250313e-16

julia> eps(1000.)
1.1368683772161603e-13

julia> eps(1e-27)
1.793662034335766e-43

julia> eps(0.0)
5.0e-324
```

Расстояние между двумя соседними представляемыми числами с плавающей точкой не является постоянным, оно меньше для меньших значений и больше для больших значений. Другими словами, представляемые числа с плавающей запятой наиболее плотно расположены на линии вещественных чисел вблизи нуля и становятся более редкими экспоненциально по мере удаления от нуля. По определению, eps(1.0) – это то же самое, что eps(Float64), поскольку 1.0 – это 64-битное значение с плавающей точкой.

Если число не имеет точного представления с плавающей точкой, оно будет округлено до соответствующего представляемого значения. По умолчанию Julia использует режим округления, называемый RoundNearest. Он округляет до ближайшего целого числа, а ничьи округляются до ближайшего четного целого числа.

Базовые операторы Julia

Следующие арифметические операторы поддерживаются для всех примитивных числовых типов:

Выражение	Наименование	Описание
+x	Унарный плюс	Операция тождества.
-x	Унарный минус	Отображает значения в их аддитивные инверсии.
x+y	Бинарный плюс	Выполняет сложение.
x-y	Бинарный минус	Выполняет вычитание.
x*y	Умножение	Выполняет умножение.
x/y	Деление	Выполняет деление.
x÷y	Целочисленное деление	Обозначается как x/y и усекается до целого числа.
x\y	Обратное деление	Эквивалентно y/x.
x^y	Возведение в степень	Возводит x в степень y.
x%y	Остаток	Остаток от евклидова деления, результат того же знака, что и x, и меньшее по величине, чем y.

Несколько простых примеров с использованием арифметических операторов:

```
julia> 1+10-5
6

julia> 2-6
–4

julia> 5*20/10
10.0

julia> 20\10
0.5

julia> 3^3 27

julia> 5.5%-2
1.5
```

Число, помещенное непосредственно перед идентификатором или круглыми скобками, например, 2x или 2(x+y), рассматривается как умножение.

Логические операторы

Для типов Bool поддерживаются следующие логические операторы:

Функция	Описание
!x	Отрицание, логическое НЕ.
x && y	Конъюнкция, логическое И.
x || y	Дизъюнкция, логическое ИЛИ.

Примеры использования логических операторов:

julia> !true
false

julia> !false
true

julia> true && true
true

julia> true && false
false

julia> false && false
false

julia> true || true
true

julia> true || false
true

julia> false || false
false

Побитовые операторы

Следующие побитовые операторы поддерживаются для всех примитивных целочисленных типов:

Выражение	Наименование
~x	Побитовое НЕ
x&y	Побитовое И
x|y	Побитовое ИЛИ
x⊻y	Побитовое исключающее ИЛИ
x >>> y	Логический сдвиг вправо
x >> y	Арифметический сдвиг вправо
x << y	Логический/Арифметический сдвиг влево

```
julia> ~100
–101

julia> 121 & 232
104

julia> 121 | 232
249

julia> 121 ⊻ 232 #Знак юникода
145

julia> xor(121, 232)
145

julia> ~UInt32(121)
0xffffff86

julia> ~UInt8(121)
0x86
```

Операторы обновления

Каждый арифметические и побитовый оператор имеет обновляющую версию, которую можно сформировать, поставив знак равенства (=) сразу после оператора. Этот оператор обновления присваивает результат операции обратно своему левому операнду.

Версии всех двоичных арифметических и побитовых операторов:

Выражение	Эквивалент
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x \= y`	`x = x \ y`
`x ÷= y`	`x = x ÷ y`
`x %= y`	`x = x % y`
`x ^= y`	`x = x ^ y`
`x &= y`	`x = x & y`
`x |= y`	`x = x | y`
`x ⊻= y`	`x = x ⊻ y`
`x >>>= y`	`x = x >>> y`
`x >>= y`	`x = x >> y`
`x <<= y`	`x = x << y`

Пример использования операторов обновления:

```
julia> x = 25
25

julia> x += 25
50

julia> x
50
```

Оператор обновления переопределяет переменную в левой части. В результате тип переменной может измениться:

```
julia> x = 0x01
0x01
```

```
julia> typeof(x)
UInt8

julia> x *= 2
2

julia> typeof(x)
Int64
```

Векторизированные “точечные” операторы

Для каждого бинарного оператора существует соответствующий "точечный" оператор, который применяет оператор поэлементно над многоэлементными структурами (массивы и т.п.).

Примеры использования “точечного” оператора:

```
julia> [2, 4 ,6].^2
3-element Vector{Int64}:
4
16
36

julia> x=[1 2 3 4 5 ; 6 7 8 9 10]
2×5 Matrix{Int64}:
1 2 3 4 5
6 7 8 9 10

julia> x.+1
2×5 Matrix{Int64}:
2 3 4 5 6
7 8 9 10 11

julia> x
2×5 Matrix{Int64}:
1 2 3 4 5
6 7 8 9 10

julia> x .+=1
2×5 Matrix{Int64}:
2 3 4 5 6
7 8 9 10 11

julia> x
2×5 Matrix{Int64}:
2 3 4 5 6
7 8 9 10 11
```

Обратите внимание, "точечные" операторы обновления изменяют исходные значения, в отличии от остальных, создающих новые.

Конец ознакомительного фрагмента.