Лабораторная работа № 1. Julia. Установка и настройка. Основные принципы.

1.1. Цель работы

Основная цель работы — подготовить рабочее пространство и инструментарий для работы с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.

1.2. Задание

1. Установите под свою операционную систему Julia, Jupyter (разделы 1.3.1 и 1.3.2).
2. Используя Jupyter Lab, повторите примеры из раздела 1.3.3.
3. Выполните задания для самостоятельной работы (раздел 1.3.4).

1.3. Последовательность выполнения работы

1.3.1. Подготовка инструментария к работе

Установите Julia (`https://julialang.org/`) и Jupyter (`https://jupyter.org/`) под вашу операционную систему.

Для ОС типа Windows рекомендуется для установки использовать менеджер пакетов Chocolatey (https://chocolatey.org/), устанавливаемый через Administrative Shell. Далее рекомендуется посредством данного менеджера установить Far Manager, Notepad++, Julia, Anaconda Distribution (Python 3.x).

После установки запустите Julia в режиме REPL (read-eval-print loop). Если вы работаете под ОС Linux, то в командной строке введите `julia`. После запуска Julia вы попадёте в режим командной строки.

Установите пакеты для работы с Jupyter. Для этого перейдите в пакетный режим Julia, нажав на клавиатуре знак закрывающейся квадратной скобки `]`, затем введите `add IJulia`.

```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.5.0 (2020-08-01)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

(@v1.5) pkg> add IJulia
```

Рис. 1.1. Установка пакетов для работы с Jupyter

В ОС Linux пакеты будут установлены в каталог `~/.local/share/jupyter/kernels` в подкаталог с соответствующей версией ядра Julia. В случае изменения версии ядра

необходимо будет использовать пакеты под новое ядро, а каталог с предыдущим номером версии ядра нужно удалить.

Для возвращения в основной командный режим Julia необходимо нажать BackSpace. Для выхода из командного интерфейса Julia используйте сочетание клавиш Ctrl + d.

Для интерактивной работы с Julia удобно использовать или Jupyter Notebook, или Jupyter Lab. По своей сути блокнот Jupyter позволяет объединить в единый документ тест, программный код, результат его выполнения и визуализацию результата. Основная работа с блокнотами осуществляется посредством браузера. Формат документов (`.ipynb`) идентичен в Jupyter Notebook и Jupyter Lab.

Для запуска Jupyter Notebook из-под ОС Linux в командной строке введите `jupyter notebook`. Если браузер автоматически не загрузит страницу с блокнотом Jupyter, то в строке браузера введите `http://localhost:8888/tree`. Для прекращения работы с Jupyter Notebook используйте сочетание клавиш Ctrl + c.

Для запуска Jupyter Lab из-под ОС Linux в командной строке введите `jupyter lab` или в строке браузера `http://localhost:8888/lab`. Для прекращения работы с Jupyter Lab используйте сочетание клавиш Ctrl + c.

В Windows запустить Jupyter Lab вы можете с помощью ярлыка, добавленного в меню «Пуск» после установки Anaconda.

1.3.2. Основы работы в блокноте Jupyter

Запустите Jupyter Lab.

Каждый блокнот (или консоль, или терминал, или текстовый редактор) располагается в своей вкладке в основной рабочей области. Для создания нового блокнота выберите в меню File > New, далее укажите, что именно вы хотите создать, например, Notebook, затем ядро Juliia-1.x. Для открытия существующего файла используйте стандартные пункты меню и навигатор. Каждый файл-блокнот представляет собой текстовый файл в формате JSON с описанием всего содержимого блокнота. Обычно файл имеет расширение `.ipynb` или `.ipyn`.

Основная концепция интерактивных блокнотов — это ячейка, содержащая отдельный фрагмент текста (или кода). Для написания текста в ячейке нужно в панели инструментов указать Markdown, для написания элемента кода — Code. Для изменения режимов вставки ячеек можно использовать также комбинации клавиш. Для этого нужно на активной ячейке нажать ESC, что выведет ячейку из режима редактирования и переведёт её в командный режим, в котором есть специальные сочетания клавиш для вставки / вырезания / изменения ячеек:

- a или b — создать новую ячейку соответственно выше или ниже текущей;
- x — удалить ячейку;
- z — отмена удаления ячейки;
- m — перевести ячейку в режим текста;
- y — перевести ячейку в режим набора кода.

Для выполнения кода внутри ячейки выберите эту ячейку и нажмите Shift + Enter или кнопку со значком Run на панели инструментов. Если ячейка содержит несколько строк кода, то при выполнении этой ячейки отобразится только результат последней строки (операции). Вывод результата можно подавить, завершив строку знаком «точка с запятой». Примеры по выполнению кода с простейшей операцией сложения в блокноте Jupyter приведены на рис. 1.2.

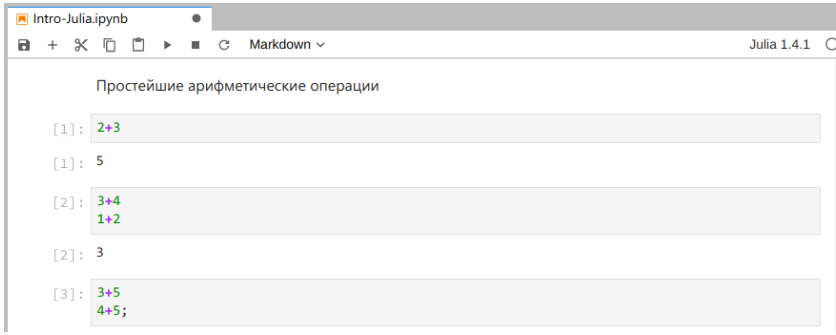

Рис. 1.2. Простейшие операции на языке Julia в Jupyter Lab

Если вам необходимо получить информацию по работе с какой-то незнакомой для вас функцией Julia, то вы можете поставить в ячейке перед названием этой функции знак вопроса (рис. 1.3).

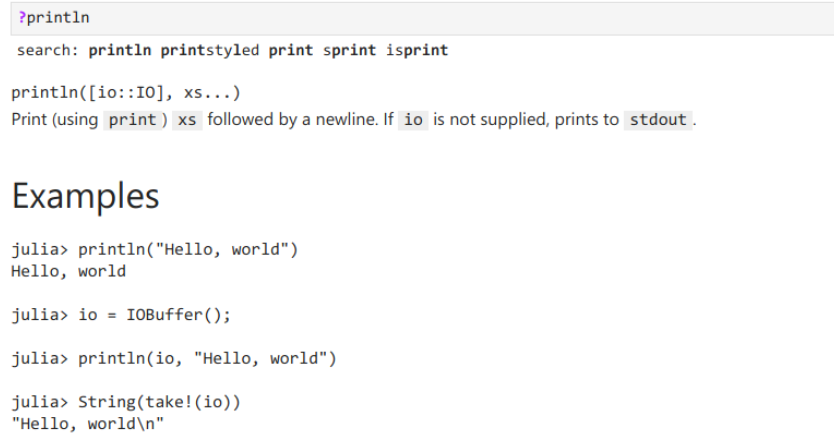

Рис. 1.3. Пример получения информации по функции println на языке Julia в Jupyter Lab

Если требуется использовать команды из командной оболочки вашей операционной системы, то перед соответствующей командой нужно поставить знак «точка с запятой». Например, для пользователей ОС Linux можно вывести текущую дату и имя пользователя, используя последовательно команды `date` и `whoami` (рис. 1.3). Для пользователей других ОС следует использовать команды оболочки соответствующей операционной системы.

Для очистки результатов выполнения ячеек следует использовать меню Edit › Clear Outputs или Edit › Clear All Outputs.

```
;date
```

```
Пн авг 17 11:27:00 MSK 2020
```

```
;whoami
```

```
akorolkova
```

Рис. 1.4. Пример получения информации о дате и пользователе ОС Linux в Jupyter Lab

1.3.3. Основы синтаксиса Julia на примерах

С основами синтаксиса языка Julia можно ознакомиться в источниках [1—5].

Далее приведены простейшие примеры с использованием синтаксиса Julia, выполненные в блокноте Jupyter Lab.

Определение типа числовой величины:

```
typeof(Number)
```

Здесь `Number` — конкретное число, например, 3 или 3.5, или числовой результат какой-либо операции, например, $3/3.5$, $\sqrt{3+4i}$, значение числа π.

В Julia введены специальные значения `Inf`, `-Inf`, `NaN`, обозначающие бесконечность и отсутствие какого-либо значения. Такие значения могут получаться в результате операций типа деления на ноль, а также могут быть допустимой частью выражений, поскольку в языке имеют тип вещественного числа (см. рис. 1.5).

Для определения крайних значений диапазонов целочисленных числовых величин можно воспользоваться следующим кодом:

```
for T in
  ↪  [Int8,Int16,Int32,Int64,Int128,UInt8,UInt16,UInt32,UInt64,UInt128]
   println("$(lpad(T,7)): [$(typemin(T)),$(typemax(T))]")
end
```

В результате получим минимальные и максимальные значения целочисленных типов (см. рис. 1.5).

В Julia преобразование типов можно реализовать или прямым указанием, например вещественное число 2.0 преобразовать в целое, а число 2 в символ:

```
Int64(2.0),  Char(2)
```

или использовать обобщённый оператор преобразования типов `convert()`, например:

```
convert(Int64, 2.0), convert(Char,2)
```

Преобразование 1 в булевое true, 0 — в булевое false:

```
Bool(1),  Bool(0)
```

Для приведения нескольких аргументов к одному типу, если это возможно, используется оператор `promote()`, например:

```
promote(Int8(1), Float16(4.5), Float32(4.1))
```

В данном выражении все аргументы оператора `promote()` в результате будут иметь тип `Float32`, в чём можно убедиться, воспользовавшись функцией определения типа `typeof()` (см. рис. 1.6).

Базовый синтаксис определения функции:

```
function <Имя> (<СписокПараметров>)
  <Действия>
end
```

Например, определим функцию $f(x)$ возведения переменной x в квадрат и возведём в квадрат число 4 (см. рис. 1.7):

```
typeof(3), typeof(3.5), typeof(3/3.55), typeof(sqrt(3+4im)), typeof(pi)
```

```
(Int64, Float64, Float64, Complex{Float64}, Irrational{:π})
```

```
1.0/0.0, 1.0/(-0.0), 0.0/0.0
```

```
(Inf, -Inf, NaN)
```

```
typeof(1.0/0.0), typeof(1.0/-0.0), typeof(0.0/0.0)
```

```
(Float64, Float64, Float64)
```

```
for T in [Int8,Int16,Int32,Int64,Int128,UInt8,UInt16,UInt32,UInt64,UInt128]
println("$(lpad(T,7)): [$(typemin(T)),$(typemax(T))]")
end
```

```
   Int8: [-128,127]
  Int16: [-32768,32767]
  Int32: [-2147483648,2147483647]
  Int64: [-9223372036854775808,9223372036854775807]
 Int128: [-170141183460469231731687303715884105728,170141183460469231731687303715884105727]
  UInt8: [0,255]
 UInt16: [0,65535]
 UInt32: [0,4294967295]
 UInt64: [0,18446744073709551615]
UInt128: [0,340282366920938463463374607431768211455]
```

Рис. 1.5. Примеры определения типа числовых величин

```
Int64(2.0), Char(2), typeof(Char(2))
```

```
(2, '\x02', Char)
```

```
convert(Int64, 2.0), convert(Char,2)
```

```
(2, '\x02')
```

```
typeof(promote(Int8(1), Float16(4.5), Float32(4.1)))
```

```
Tuple{Float32,Float32,Float32}
```

Рис. 1.6. Примеры приведения аргументов к одному типу

```
function f(x)
  x^2
end
f(4)
```

Другой способ определения несложных функций:

```
<Имя> (<СписокПараметров>) = <Выражение>
```

Например (см. рис. 1.7):

```
g(x)=x^2
```

Пример определения одномерных массивов (вектор-строка и вектор-столбец) и обращение к их вторым элементам:

```
a = [4 7 6] # вектор-строка
b = [1, 2, 3] # вектор-столбец
a[2], b[2] # вторые элементы векторов a и b
```

Пример определения двумерного массива (матрицы) и обращение к его элементам:

```
function f(x)
    x^2
end
```

```
f (generic function with 1 method)
```

```
f(4)
```

```
16
```

```
g(x)=x^2
```

```
g (generic function with 1 method)
```

```
g(8)
```

```
64
```

Рис. 1.7. Примеры определения функций

```
a = 1; b = 2; c = 3; d = 4 # присвоение значений
Am = [a b; c  d] # матрица 2 x 2
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы
```

Пример выполнения операций над массивами (`aa'` — транспонирование вектора (рис. 1.8)):

```
aa = [1 2]
AA = [1 2; 3 4]
aa*AA*aa'
```

```
a = [4 7 6] # вектор-строка
b = [1, 2, 3] # вектор-столбец
a[2], b[2] # вторые элементы векторов  a и b
```

```
(7, 2)
```

```
a = 1; b = 2; c = 3; d = 4 # присвоение значений
Am = [a b; c  d] # матрица 2 x 2
```

```
2×2 Array{Int64,2}:
 1  2
 3  4
```

```
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы Am
```

```
(1, 2, 3, 4)
```

```
aa = [1 2]  # вектор-строка
AA = [1 2; 3 4] # матрица 2 x 2
aa*AA*aa' # умножение вектора=строки на матрицу и на вектор-столбец (операция транспонирования)
```

```
1×1 Array{Int64,2}:
 27
```

```
aa, AA, aa'
```

```
([1 2], [1 2; 3 4], [1; 2])
```

Рис. 1.8. Примеры работы с массивами

1.3.4. Задания для самостоятельной работы

1. Изучите документацию по основным функциям Julia для чтения / записи / вывода информации на экран: `read()`, `readline()`, `readlines()`, `readdlm()`, `print()`, `println()`, `show()`, `write()`. Приведите свои примеры их использования, поясняя особенности их применения.
2. Изучите документацию по функции `parse()`. Приведите свои примеры её использования, поясняя особенности её применения.
3. Изучите синтаксис Julia для базовых математических операций с разным типом переменных: сложение, вычитание, умножение, деление, возведение в степень, извлечение корня, сравнение, логические операции. Приведите свои примеры с пояснениями по особенностям их применения.
4. Приведите несколько своих примеров с пояснениями с операциями над матрицами и векторами: сложение, вычитание, скалярное произведение, транспонирование, умножение на скаляр.

1.4. Содержание отчёта

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка задания работы.
3. Описание выполнения задания:
   - подробное пояснение выполняемых в соответствии с заданием действий;
   - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
   - листинги (исходный код) программ и результаты его выполнения;
4. Выводы, согласованные с заданием работы.

Список литературы

1. Julia 1.5 Documentation. — 2020. — URL: `https://docs.julialang.org/en/v1/`.
2. *Klok H., Nazarathy Y.* Statistics with Julia: Fundamentals for Data Science, Machine Learning and Artificial Intelligence. — 2020. — URL: `https://statisticswithjulia.org/`.
3. *Ökten G.* First Semester in Numerical Analysis with Julia. — Florida State University, 2019. — DOI: `10.33009/jul`.
4. *Антонюк В. А.* Язык Julia как инструмент исследователя. — М. : Физический факультет МГУ им. М. В. Ломоносова, 2019.
5. *Шиндин А. В.* Язык программирования математических вычислений Julia. Базовое руководство. — Нижний Новгород : Нижегородский госуниверситет, 2016.