Библиотеки

Установка и подключение библиотек

Введение

Библиотеки Julia – это организованный набор готовых функций, классов и объектов для решения различных задач. Библиотеки **Julia** распространяются в виде пакетов, которые можно установить с помощью менеджера пакетов Pkg и использовать в своих проектах.

В **Engee** работа с библиотеками **Julia** также осуществляется через встроенный менеджер пакетов Pkg с помощью двух инструментов рабочего пространства – командную строку и редактор скриптов.

При работе с большим количеством библиотек удобно использовать редактор скриптов, в котором после вызова Pkg не требуется его повторная инициализация. Командная строка удобна в том случае, если вам привычнее работать с кодом вне интерфейса редактора скриптов и вы работаете с небольшим количеством библиотек.

Принцип работы с библиотеками

Найдите точное название интересующей вас библиотеки. Экосистема **Julia** содержит более 10 000 пакетов, зарегистрированных в общем реестре, что может сильно затруднить поиск нужного пакета. Для оптимизации поиска рекомендуем обратиться к следующим источникам:

- Внешние библиотеки – раздел документации **Engee** с описанием большого числа библиотек.
- JuliaHub – сервис для поиска по всей зарегистрированной документации пакетов с открытым исходным кодом с возможностью сортировки по тегам и ключевым словам.
- Julia Packages – онлайн-ресурс, предназначенный для поиска, изучения и просмотра пакетов **Julia** с возможностью настройки фильтров по категориям, популярности и датам.
- Julia.jl – главный репозиторий пакетов для языка программирования **Julia**. Этот репозиторий хранится на платформе **GitHub** и содержит множество полезных пакетов и инструментов, разработанных сообществом для работы с языком **Julia**.

Выберите инструмент для работы с библиотеками – командная строка или редактор скриптов в зависимости от ваших задач.

Инициализируйте работу с пакетами в **Julia**, добавив менеджер пакетов Pkg. Синтаксис добавления менеджера отличается в зависимости от выбранного инструмента.

Добавьте библиотеки с помощью команды add в ваше окружение, чтобы использовать их в своем проекте. Синтаксис добавления библиотек зависит от выбранного инструмента.

Загрузите библиотеку в пространство имен и получите доступ к ее конкретным элементам с помощью команд `using` и `import`. Вы также можете добавить и собственные библиотеки.

При необходимости узнайте статус библиотек через `status`, проверьте, какие библиотеки загружены в оперативную память **Engee**, и удалите ненужные с помощью команд `remove` или `rm`. Синтаксис зависит от выбранного инструмента.

Библиотеки Julia в командной строке

Чтобы установить новую библиотеку **в командной строке Engee**, кликните левой кнопкой мыши в области ввода кода и нажмите закрывающую квадратную скобку `]`. Это переключит командную строку на работу с менеджером пакетов. Строка изменит свой вид с engee> на pkg>:

```
pkg>
```

В режиме менеджера пакетов используем команду add для добавления библиотеки:

```
add Example
```

Вы можете добавить несколько библиотек одновременно, перечисляя их имена через запятую, например:

```
add Example, Plots, Pluto
```

Для удаления библиотеки используется команда `remove` или ее сокращенная версия `rm`:

```
remove Pluto
```

При необходимости вы можете удалить несколько библиотек за раз, перечисляя их имена через запятую:

```
remove Pluto, Plots, Example
```

Для выхода из режима менеджера пакетов pkg> кликните левой кнопкой мыши по области ввода кода и нажмите Backspace. Это вернет вас в первичный вид командной строки engee>.

Библиотеки Julia в редакторе скриптов

Чтобы установить новую библиотеку **в интерактивном скрипте Engee**, в секции с кодом импортируйте менеджер пакетов Pkg командой `import Pkg`, а затем выполните команду `Pkg.add("Имя библиотеки")`. Например, секция с кодом для добавления библиотеки Example будет выглядеть так:

[ ]:
```
import Pkg
Pkg.add("Example")
```

После запуска этой ячейки будет импортирован менеджер пакетов Pkg и будет установлена библиотека Example. Информация о ходе установки библиотеки отобразится автоматически:

```
Resolving package versions...
Installed Example — v0.5.3
Updating `/user/start/Project.toml`
[7876af07] + Example v0.5.3
Updating `/user/start/Manifest.toml`
[7876af07] + Example v0.5.3
```

Как и в случае с командной строкой, вы можете добавить несколько библиотек одновременно, перечисляя их имена через запятую и добавляя квадратные скобки для создания массива строк, например:

```
Pkg.add(["Example", "Plots"])
```

Для удаления библиотеки используется команда rm:

[ ]:
```
Pkg.rm("Example")
```

Для удаления нескольких библиотек одновременно используется перечисление их имен через запятую и квадратные скобки:

```
Pkg.rm(["Example", "Plots"])
```

Команда remove не определяется в редакторе скриптов, хотя и используется в командной строке. Попытка вызвать команду remove в редакторе скриптов выдаст ошибку.

В редакторе скриптов имя менеджера пакетов Pkg и имена библиотек **Julia** чувствительны к регистру и должны начинаться с заглавной буквы, в противном случае вы получите сообщение об ошибке.

✏Задание 1

В редакторе скриптов установите библиотеку StatsPlots.jl, предназначенную для построения статистических графиков.

[ ]:
```

```

Подсказка Сначала импортируйте менеджер пакетов Pkg с помощьюю команды `import Pkg`. Затем установите библиотеку с помощью команды `Pkg.add("Имя библиотеки")`. Имя библиотеки запишите в двойных кавычках. Расширение `.jl` писать не нужно.

Решение

[ ]:
```
import Pkg
Pkg.add("StatsPlots")
```

Статус библиотек

Чтобы узнать, какие библиотеки установлены и готовы к использованию, откройте **командную строку** и выполните следующие действия:

1. Войдите в менеджер пакетов Pkg через `]`.
2. Введите команду `status`:

```
pkg> status
```

После этого вы увидите список установленных библиотек и их версию. Например, так:

```
pkg> status
Status `/user/.project/Project.toml`
⌅ [c3fe647b] AbstractAlgebra v0.27.8
⌅ [7d9f7c33] Accessors v0.1.27
⌅ [79e6a3ab] Adapt v3.5.0
⌅ [91a5bcdd] Plots v1.36.1
^ [c3e4b0f8] Pluto v0.19.36
```

Символы ^ и ⌅ означают, что для библиотек есть новые версии:

- ^ – имеется более новая версия.
- ⌅ – имеется более новая версия, но есть конфликты по совместимости с другими библиотеками.

Узнать какие библиотеки установлены и готовы к использованию можно и через **редактор скриптов**:

1. Введите `import Pkg`, если не сделали этого ранее.
2. Введите `Pkg.status()` для получения статуса библиотек:

[ ]:
```
Pkg.status()
```

Загрузка библиотек

В **Julia** для загрузки библиотек и предоставления доступа к их конкретным элементам, таким как функции, типы и переменные, используются операторы `import` и `using`.

Оба оператора позволят вам использовать функции из библиотек или загружать их, но есть небольшая разница в том, как они экспортируются в текущее пространство имен:

- `using` – экспортирует все функции.
- `import` – требует явного указания, какие функции нужно импортировать.

Например:

```
[ ]: import Base.Math.cos
     x = cos(0.5)
```

В случае с `import` перед вызовом функции необходимо писать название библиотеки и ставить точку `.`.

Этот код импортирует функцию `cos` из библиотеки `Base.Math` и вызывает эту функцию с аргументом 0.5. Функция `cos` вычисляет косинус угла в радианах и присваивает это значение переменной `x`. Таким образом, переменная будет содержать результат вычисления косинуса угла 0.5 радиан, равный 0.8775825618903728.

Оператор `using` предназначен для загрузки библиотек целиком, а не отдельных функций. Тогда код с использованием оператора `using` будет иметь вид:

```
[ ]: using Base.Math
     y = cos(0.5)
```

Таким образом, `using` используется для загрузки библиотек и автоматического экспорта всех ее выбранных имен в текущее пространство имен, обеспечивая удобный доступ к функциям и переменным модуля. `import` также загружает библиотеку, но не экспортирует ее имена автоматически, что позволяет избежать возможных конфликтов имен и явно управлять импортированными элементами. Выбирайте операторы в зависимости от ваших задач.

✏Задание 2

Загрузите библоитеку `StatsPlots.jl`. Затем используйте функцию `scatter(x,y)` из этой библиотеки, чтобы построить точечный график для следующих данных: `x=[1 2 3 4 5]` и `y=[8 10 14 16 20]`.

```
[ ]:
```

Подсказка Для загрузки библоитеки используйте оператор `using`. Затем задайте два вектора `x` и `y`, записывая их элементы через пробел в квадратных скобках. Затем постройте точечный график с помощью функции `scatter(x,y)`.

Решение

```
[ ]: using StatsPlots
     x = [1 2 3 4 5]
     y = [8 10 14 16 20]
     scatter(x, y)
```

Библиотеки в оперативной памяти

Чтобы узнать, какие библиотеки загружены в оперативную память (скомпилированы), выполните следующие действия:

1. Откройте командную строку или редактор скриптов.
2. Введите `Base.loaded_modules` и нажмите `Enter`:

```
Base.loaded_modules
```

После этого на экране появится список библиотек, загруженных в оперативную память. Например, такой:

```
engee> Base.loaded_modules
Dict{Base.PkgId, Module} with 455 entries:
  IOCapture [b5… => IOCapture
  StatsFuns [4c… => StatsFuns
  Combinatorics… => Combinatorics
  MPFR_jll [3a9… => MPFR_jll
  OpenSSL_jll [… => OpenSSL_jll
  Xorg_xcb_util… => Xorg_xcb_util_jll
  ArrayLayouts … => ArrayLayouts
  libass_jll [0… => libass_jll
  CSV [336ed68f… => CSV
  Qt5Base_jll [… => Qt5Base_jll
  Xorg_xcb_util… => Xorg_xcb_util_image_jll
  ZipFile [a539… => ZipFile
  EllipsisNotat… => EllipsisNotation
  Libgcrypt_jll… => Libgcrypt_jll
```

`Base.loaded_modules` – это словарь в **Julia**, содержащий информацию о загруженных библиотеках в текущем сеансе. Ключи в этом словаре – это символы, представляющие имена модулей, а значения – соответствующие модули.

Загруженные в оперативную память библиотеки – это библиотеки, для которых код был загружен и скомпилирован в оперативной памяти **Engee** и готов к использованию. Это полезно, потому что вы можете работать с функциями и возможностями этих библиотек в своем коде без необходимости повторной загрузки или компиляции. Однако, чтобы использовать библиотеки в своем коде, вам все равно нужно указать с помощью команд `import` или `using`, что вы планируете их использовать.

Обзор основных библиотек Julia

Ниже приведен краткий обзор некоторых часто используемых библиотек **Julia**.

Визуализация

- `Plots.jl` – основная библиотека, предназначенная для визуализации данных (для построения различных видов графиков по числовым данным).
- `PlotlyJS.jl` – библиотека для визуализации данных. Предоставляет дополнительные возможности построения интерактивных графиков и сохранения графиков в файлах.

Системные библиотеки

- `Matlab.jl` – позволяет вызывать функции **MATLAB** из **Julia**.
- `PyCall.jl` – позволяет вызывать функции языка **Python** из **Julia**.

Работа с файловыми форматами

- `MAT.jl` – библиотека предоставляет инструменты для чтения и записи файлов данных формата **MATLAB** в **Julia**.
- `JLD2.jl` – библиотека позволяет сохранять и загружать структуры данных **Julia** в формате, представляющем собой подмножество стандарта HDF5.
- `XLSX.jl` – библиотека для чтения и записи файлов электронных таблиц **Excel**.
- `CSV.jl` – библиотека для работы с текстовыми данными с разделителями, будь то разделитель-запятая (CSV), разделитель-символ табуляции (TSV) или иной символ.
- `FileIO.jl` – библиотека предоставляет общий фреймворк для определения форматов файлов и диспетчеризации в соответствующие методы чтения и записи. Две основные функции в этом пакете называются `load` и `save`. Они обеспечивают высокоуровневую поддержку форматированных файлов, в отличие от низкоуровневых функций `read` и `write` в **Julia**.

Математические библиотеки

- `LinearAlgebra.jl` – библиотека содержит множество функций для решения задач линейной алгебры.
- `Roots.jl` – библиотека содержит функции для решения уравнений вида $f(x) = 0$.
- `DifferentialEquations.jl` – библиотека для численного решения обыкновенных дифференциальных уравнений.
- `NumericalIntegration.jl` – библиотека для численного интегрирования дискретизированных данных.
- `ControlSystems.jl` – библиотека содержит функции для анализа и синтеза систем управления (в первую очередь линейных).
- `Interpolations.jl` – библиотека содержит функции для различных методов интерполяции.
- `DynamnicalSystems.jl` – библиотека содержит функции для решения задач нелинейной динамики и анализа временных рядов.
- `Calculus.jl` – библиотека содержит функции для базовых операций дифференцирования и интегрирования.
- `GLPK.jl` – библиотека содержит функции линейного програмирования и линейной оптимизации.
- `Ipopt.jl` – библиотека для нелинейной оптимизации задач большой размерности.
- `Distributions.jl` – библиотека содержит функции для различных распределений вероятности.
- `Polynomials.jl` – библиотека обеспечивает основные арифметические действия, интегрирование, дифференцирование, вычисление значений, нахож-

дение корней и аппроксимацию для одномерных многочленов.
- TaylorSeries.jl – библиотека содержит функции для разложения в ряд Тейлора функций одной или двух переменных.
- NLSolve.jl – библиотека содержит функции для решения систем нелинейных уравнений.
- Symbolics.jl – библиотека для символьных вычислений.
- Flux.jl – библиотека для машинного обучения.

Типы данных

- DataFrames.jl – библиотека предоставляет инструментарий для работы с табличными данными.
- LaTeXStrings.jl – библиотека, упрощающая ввод уравнений **LaTeX** в переменных и константах строкового типа.

Обработка сигналов и связь

- FFTW.jl – библиотека содержит функции для быстрого преобразования Фурье.
- Noise.jl – библиотека содержит функции для добавления различных видов шума к сигналам и изображениям.
- DSP.jl – библиотека содержит ряд функций для цифровой обработки сигналов (расчет периодограмм, оконные функции, синтез фильтров и фильтрация, свертка, корреляция, кодирование с линейным прогнозированием).
- DigitalComm.jl – библиотека предоставляет некоторые полезные инструменты для управления блоками цифровой связи (манипуляция с битами, модуляция и демодуляция, генерация и декодирование сигналов с несколькими несущими).

Обработка изображений

ImageBinarization.jl – библиотека содержит ряд алгоритмов для анализа и бинаризации изображений.

Статистика

- StatsBase.jl – библиотека обеспечивает решение базовых задач статистики.
- HypothesisTests.jl – библиотека содержит различные функции для проверки статистических гипотез.
- GLM.jl – библиотека для построения линейных и обобщенных линейных моделей.

Теперь рассмотрим более подробно возможности некоторых из этих библиотек.

Библиотека `Plots.jl`

Библиотека `Plots.jl` представляет собой интерфейс и набор инструментов для визуализации данных.

Эта библиотека уже предустановлена в **Engee**, поэтому начать использовать ее можно с помощью следующей команды:

[ ]:
```
using Plots
```

Рассмотрим некоторые возможности этого пакета. Наиболее универсальный инструмент, служащий для построения графиков по числовым данным, – функция `plot`.

Пример. Построим график функци синус. Для координат x можно создать диапазон от 0 до 10, состоящий, скажем, из 100 элементов. Для координат y можно создать вектор, вычислив `sin(x)` поэлементно. Для этого мы пишем точку сразу после вызова функции. Наконец, мы используем функцию `plot` для построения линии.

[ ]:
```
x = range(0, 10, length=100)
y = sin.(x)
plot(x, y)
```

В `Plots.jl` каждый столбец представляет собой **ряд** – набор связанных точек, образующих линии, поверхности или другие примитивы построения графиков. Мы можем построить несколько линий, задав матрицу значений, где каждый столбец интерпретируется как отдельная линия. Ниже `[y1 y2]` формирует матрицу размером 100×2 (100 строк, 2 столбца).

[ ]:
```
x = range(0, 10, length=100)
y1 = sin.(x)
y2 = cos.(x)
plot(x, [y1 y2])
```

Кроме того, можно добавлять на полотно дополнительные линии, изменяя объект `plot`. Для этого используется команда `plot!`, где `!` обозначает, что команда изменяет текущий график.

[ ]:
```
y3 = sin.(x).^2 .- 1/2
plot!(x, y3)
```

✏Задание 3

С помощью функций `plot` и `plot!` постройте на одном полотне графики функций $y = x^2$ и $y = x^3$, задав значения x в виде диапазона от -10 до 10, состоящего из 100 элеменнтов.

[ ]:
```

```

Решение

[ ]:
```
using Plots
x = range(-10, 10, length=100)
y1 = x.^2
y2 = x.^3
plot(x, y1)
plot!(x, y2)
```

Сохранение графиков выполняется с помощью команды `savefig`. Например:

[ ]:
```
savefig("myplot.png")
```

Графики в **Julia** можно стилистически оформить. В `Plots.jl` модификаторы графиков называются **атрибутами**. При работе с данными и атрибутами соблюдаются следующите правила: позиционные аргументы соответствуют входным данным, а именованные аргументы соответствуют атрибутам.

Так, например, `plot(x, y, z)` – это трехмерные данные для трехмерных графиков без атрибутов, а `plot(x, y, attribute=value)` – это двумерные данные с одним атрибутом, которому присвоено некоторое значение.

В качестве примера рассмотрим изменение ширины линии с помощью `linewidth` (или его псевдонима `lw`), изменение надписей условных обозначений с помощью `label` и добавление названия с помощью `title`. Применим все это к графикам синуса и косинуса:

[ ]:
```
x = range(0, 10, length=100)
y1 = sin.(x)
y2 = cos.(x)
plot(x, [y1 y2], title="Trigonometric functions", label=["sin(x)"␣
  ↪"cos(x)"], linewidth=3)
```

Еще одним широко используемым видом графиков является график рассеяния (точки, не соединенные линиями). Он строится с помощью функции `scatter`. Например, построим график синуса и добавим к синусу случайную составляющую.

[ ]:
```
x = range(0, 10, length=100)
y = sin.(x)
y_noisy = y .+ 0.1*randn(100)
plot(x, y, label="sin(x)")
scatter!(x, y_noisy, label="data")
```

Больше о библиотеке `Plots.jl` вы можете узнать в документации.

Библиотека `Roots.jl`

`Roots.jl` – это библиотека **Julia** для нахождения нулей непрерывных скалярных функций одной вещественной переменной с использованием чисел с плаваю-

щей запятой. То есть она служит для решения уравнений вида $f(x) = 0$ с учетом специфических особенностей чисел с плавающей запятой.

Установим и подключим библиотеку `Roots.jl`:

[ ]:
```
import Pkg
Pkg.add("Roots")
using Roots
```

Основной интерфейс предоставляет функция `find_zero`. Она поддерживает различные алгоритмы путем спецификации метода.

Пример. Известно, что для функции $F(x) = x^5 - x + 1/2$ один ноль находится на отрезке между –1,2 и –1,0, а еще два – возле 0,6.

Для нуля между двумя значениями, при которых функция меняет знак, будет полезен ограничивающий метод, так как согласно теореме о промежуточном значении такие методы гарантированно сходятся для непрерывных функций. Ограничивающий алгоритм используется, если начальные приближения передаются в виде кортежа `(-1.2, -1)`:

[ ]:
```
F(x) =  x^5 - x + 1/2
find_zero(F, (-1.2,  -1))
```

Для нулей «в окрестности» точки часто применяются неограничивающие методы, так как в общем случае такие алгоритмы эффективнее и могут использоваться в случаях, когда ноль не пересекает ось Ox. Поэтому мы можем задать только начальное приближение:

[ ]:
```
find_zero(F,  0.6)
```

Однако все вещественные корни функции $f(x)$ находятся между –5 и 5. Функция `find_zeros` использует эвристические процедуры и ряд алгоритмов для нахождения всех нулей в заданном диапазоне. В данном случае метод успешно находит все 3 корня:

[ ]:
```
find_zeros(F, -5,  5)
```

✏Задание 4

С помощью функции `find_zero` найдите оба корня уравнения $x^2 - 5 = 0$, зная, что они находятся в окрестностях точек $x = 2$ и $x = -2$.

[ ]:
```

```

Решение

[ ]:
```
using Roots
F(x) = x^2 - 5
find_zero(F, 2)
```

[ ]:
```
find_zero(F, -2)
```

Больше о библиотеке `Roots.jl` вы можете узнать в документации.

Библиотека `Symbolics.jl`

`Symbolics.jl` – это библиотека символьных вычислелний.

Для объявления символьных переменных используется макрос `@variables`:

[ ]:
```
using Symbolics
@variables x y
```

После объявления символьных переменных можно генерировать символьные выражения. Например:

[ ]:
```
z = x^2 + y
```

Можно создать оператор дифференцирования с помощью `Differential`:

[ ]:
```
z = x + x^2
D = Differential(x)
D(z)
```

Заметьте, что этот код еще не вычисляет производную: `D` – это ленивый оператор, так как позволяет символически представить производную y по x, что полезно, например, при представлении дифференциальных уравнений. Однако если мы хотим получить символьное выражение для производной, будем использовать `expand_derivatives`:

[ ]:
```
expand_derivatives(D(z))
```

Для упрощения символьных выражений используется команда `Symbolics.simplify`:

[ ]:
```
Symbolics.simplify(2x + 3x + 5x)
```

✏Задание 5

Объявите символьные переменные `a` и `b`. Затем с помощью функции `Symbolics.simplify` упростите выражение $(a^2 - b^2)/(a - b)$.

[ ]:
```

```

Решение

[ ]:
```
using Symbolics
@variables a b
```

```
Symbolics.simplify((a^2-b^2)/(a-b))
```

Больше о библиотеке `Symbolics.jl` вы можете узнать в документации.

Библиотека `NumericalIntegration.jl`

Библиотека `NumericalIntegration.jl` – это простой пакет, предназначенный для численного интегрирования предварительно дискретизированных данных (то есть вы не можете выбирать произвольные узлы).

Пример. Интегрирование методом трапеций (по умолчанию) с помощью функции `integrate`:

[ ]:
```
using NumericalIntegration
x = collect(-π/2 : π/1000 : π/2)
y = sin.(x)
integrate(x, y)
```

Интегрирование методом Симпсона:

[ ]:
```
integrate(x, y, SimpsonEven())
```

Интегрирование с накоплением:

[ ]:
```
Y = cumul_integrate(x, y)
```

Больше о библиотеке `NumericalIntegration.jl` вы можете узнать в документации.

Если у вас есть возможность задавать подынтегральную функцию в произвольных точках, используйте более эффективную библиотеку `FastGaussQuadrature.jl`.

Библиотека `LinearAlgebra.jl`

Библиотека `LinearAlgebra.jl` содержит реализацию множества часто используемых операций линейной алгебры.

Пример. Для матрицы A вычислим след, определитель и обратную матрицу с помощью функций `tr`, `det` и `inv` соответственно.

[ ]:
```
using LinearAlgebra
A = [1 2 3; 4 1 6; 7 8 1]
```

[ ]:
```
tr(A)
```

[ ]:
```
det(A)
```

[ ]:
```
inv(A)
```

Собственные значения и собственные векторы матрицы находятся с помощью функций `eigvals` и `eigvecs` соответственно:

[ ]:
```
A = [-4. -17.; 2. 2.]
eigvals(A)
```

[ ]:
```
eigvecs(A)
```

Кроме того, в библиотеке `LinearAlgebra.jl` доступно множество разложений матриц. Например, LU-разложение выполняется с помощью функции `factorize`:

[ ]:
```
A = [1.5 2 -4; 3 -1 -6; -10 2.3 4]
factorize(A)
```

✏Задание 6

Задайте матрицу $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}$. С помощью функций библиотеки `LinearAlgebra.jl` для этой матрицы вычислите след, определитель, обратную матрицу, собственные значения и собственные векторы.

[ ]:
```

```

Решение

[ ]:
```
using LinearAlgebra
A = [1 2 3; 4 5 6; 7 8 10]
tr(A)
```

[ ]:
```
det(A)
```

[ ]:
```
inv(A)
```

[ ]:
```
eigvals(A)
```

[ ]:
```
eigvecs(A)
```

Больше о библиотеке `LinearAlgebra.jl` вы можете узнать в документации.

Создание собственных библиотек

Для создания собственных библиотек используется команда `module`. Для этого нужно выполнить следующее:

1. Создадим скрипт `module` и назовем его так, как бы мы хотели назвать свою библиотеку (в нашем случае `Test_lib`) и добавим содержимое модуля, например функцию `get_value`, которая просто возвращает входное значение:

[ ]:
```
module Test_lib
    function get_value(x)
        return x
    end
end
```

2. Для загрузки созданной библиотеки используем оператор `import` или `using`.

`import.Test_lib.get_value`

или

`using.Test_lib`

3. Далее получим требуемую функцию с помощью оператора `import`:

[ ]:
```
import.Test_lib.get_value
import_value = get_value(7)
```

или с помощью оператора `using`:

[ ]:
```
using.Test_lib
using_value = get_value(7)
```

Добавление точки между `import` и именем библиотеки обязательно в случае добавления собственной библиотеки из `module`. Аналогично и для `using`.

✏Задание 7

Создайте библиотеку `My_lib`, содержащую одну функцию `opp(x)`, которая возвращает число, противоположное входному значению. Затем загрузите функцию `opp` из библиотеки `My_lib` с помощью оператора `import.My_lib.opp`. После этого с помощью функции `opp` найдите число, противоположное числу 5.

[ ]:
```

```

Решение

[ ]:
```
# создаем библиотеку
module My_lib
    function opp(x)
        return -x
    end
end
```

[ ]:
```
# используем библиотеку
import.My_lib.opp
```

```
opp(5)
```

Тест для получения сертификата

Пройти тест по теме **Библиотеки**.