Глава 1. Общие сведения о языке Julia

1.1 Установка Julia и Jupyter

1.1.1 Официальный сайт Julia

- Официальный сайт языка программирования Julia можно найти по адресу https://julialang.org/.

На этом сайте можно найти всю необходимую информацию о языке, его особенности, руководство, документацию, а также новости и обновления о развитии языка.

1.1.2. Страница загрузки

- Страница для загрузки различных версий языка Julia доступна по адресу https://julialang.org/downloads/.

Здесь представлена информация о том, как скачать язык для различных операционных систем, таких как Linux, macOS и Windows.

1.1.3 Установка Julia на Linux

```
curl -fsSL https://install.julialang.org | sh
```

1.1.4. Установка Jupyter

1. Установка базового пакета `notebook`:

   ```
   pip install notebook
   ```
   Эта команда устанавливает Jupyter Notebook, который позволяет создавать и редактировать интерактивные блокноты для программирования.

2. Дополнительная установка JupyterLab (это более современная и функциональная версия Jupyter Notebook):

   ```
   pip install jupyterlab
   ```
3. Альтернативный способ установки с использованием `pipx` (утилита для изоляции установки пакетов):

```
pipx install notebook
pipx install jupyterlab
```
Вместо обычной установки через `pip`, можно использовать `pipx`, чтобы установить каждый из этих пакетов в изолированное виртуальное окружение, что поможет избежать конфликтов версий с другими пакетами.

4. Установка Jupyter с использованием пакета из репозитория:

   ```
   sudo apt install jupyter-notebook
   ```
5. Установка языкового пакета для русскоязычного интерфейса JupyterLab:

   ```
   pip install jupyterlab-language-pack-ru-RU
   ```

1.1.5. Особенности ввода в REPL (Read-Eval-Print Loop) языка Julia

Вход в REPL Julia: Для начала работы с Julia откройте терминал и введите команду julia. Это запустит интерактивную оболочку REPL (Read-Eval-Print Loop), где можно вводить и выполнять команды на языке Julia.

1. **Системная оболочка**: Режим системных команд предоставляет доступ к командной оболочке операционной системы для выполнения системных операций. Для активации этого режима введите точку с запятой **;** в начале строки.

Чтобы вернуться в основной режим, нужно нажать клавишу BackSpace.

2. **Режим справки**: В REPL доступна система помощи. Для получения справочной информации о функции или пакете можно использовать команду с вопросительным знаком **?**. Например, чтобы узнать, что делает функция **sqrt**, можно ввести:

   ```
   ?sqrt
   ```
   Это откроет справочную информацию о функции.

Чтобы вернуться в основной режим, нужно нажать клавишу BackSpace.

3. **Управления пакетами**: Для активации режима управления пакетами в REPL Julia введите символ **]**. В этом режиме можно управлять пакетами (устанавливать, обновлять, удалять).

Можно также управлять пакетами через API, импортируя модуль Pkg командой **using Pkg**, а затем вызывая команды, например, **Pkg.add("имя пакета")**.

Полезные команды диспетчера пакетов:

- **status:** Показывает список установленных пакетов с их версиями.
- **update:** Обновляет локальный индекс пакетов и устанавливает последние версии всех пакетов.
- **add <имя пакета>:** Устанавливает новый пакет. Для нескольких пакетов используйте **add <имя пакета 1> <имя пакета 2>**.
- **free <имя пакета>:** Возвращает пакет к последней стабильной версии.
- **rm <имя пакета>:** Удаляет пакет и все его зависимости.
- **add https://github.com/<имя репозитория>/<имя пакета>.jl:** Устанавливает пакет с GitHub по URL.

Чтобы вернуться в основной режим, нужно нажать клавишу BackSpace.

4. **Использование установленных пакетов**:

- using: Предоставляет прямой доступ ко всем функциям пакета.

```
using MyPackage # Прямой доступ к функциям пакета
my_function()  # Функция вызывается без указания имени пакета
```

- import: Требует использования полных имен функций пакета, помогает избежать конфликтов имен.

```
import MyPackage
# Для вызова функции нужно использовать полное имя пакета
MyPackage.my_function()  # Нужно явно указать
MyPackage.my_function
```

Список библиотек, которые используются в пособии:

1. Библиотеки для построения графиков:
   - `Plots.jl`
   - `PyPlot.jl`
   - `Graphics.jl`
2. Библиотека для создания графических интерфейсов:
   - `Gtk.jl`
3. Библиотека для функций вывода в стиле C:
   - `Printf.jl`
4. Библиотека для работы с линейной алгеброй:
   - `LinearAlgebra.jl`
5. Библиотека для интерполяции:

- **`Interpolations.jl`**

6. Библиотека для генерации случайных чисел:
   - **`Random.jl`**
7. Библиотека для работы с датами и временем:
   - **`Dates.jl`**
8. Библиотека для решения дифференциальных уравнений:
   - **`DifferentialEquations.jl`**

1.1.6. Подключение Julia к Jupyter

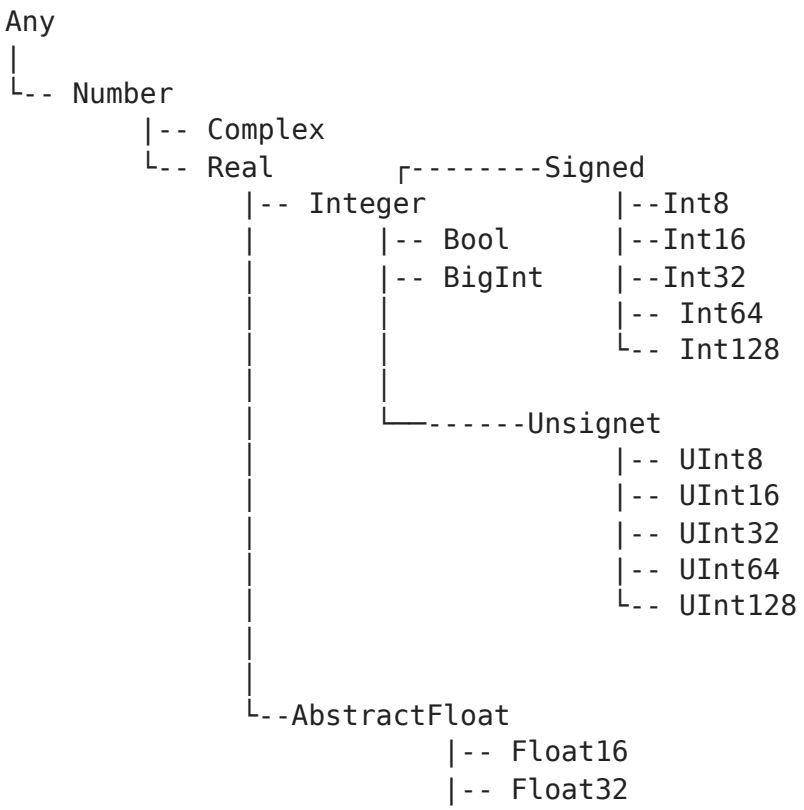
Для использования языка Julia в Jupyter необходимо установить пакет, который добавит поддержку Julia в эту среду:

```
add IJulia
```

1.2. Числовые типы данных в Julia

Иерархия числовых типовданных

Иерархия типов данных в языке программирования Julia организована таким образом, что каждый тип наследует характеристики от более общего типа. Давайте подробно рассмотрим, как устроена эта иерархия.



```
                    |-- Float16
                    └-- BigFloat
```

1. **Any** - самый общий тип в Julia, от которого наследуются все другие типы данных. Это супертип для всех типов.
2. **Number** - включает все числовые типы. Он делится на два подтипа:
   - **Complex** - тип для комплексных чисел.
   - **Real** - он включает в себя:
     - **Integer** - тип для целых чисел:
       - **Signed** - знаковые целые числа, такие как:
         - **Int8** - 8-битное целое число.
         - **Int16** - 16-битное целое число.
         - **Int32** - 32-битное целое число.
         - **Int64** - 64-битное целое число.
         - **Int128** - 128-битное целое число.
       - **Bool** - булев тип (может быть только `true` или `false`).
       - **BigInt** - произвольной точности целое число, не ограниченное стандартными размерами.
     - **Unasigned** - тип для целых чисел без знака:
       - **UInt8** - 8-битное целое число без знака.
       - **UInt16** - 16-битное целое число без знака.
       - **UInt32** - 32-битное целое число без знака.
       - **UInt64** - 64-битное целое число без знака.
       - **UInt128** - 128-битное целое число без знака.
     - **AbstractFloat** - тип для вещественных чисел:
       - **Float16** - 16-битное вещественное число.
       - **Float32** - 32-битное вещественное число.
       - **Float64** - 64-битное вещественное число.
       - **BigFloat** - произвольной точности вещественное число, не ограниченное стандартными размерами.

Теперь давайте более подробно рассмотрим каждый из типов данных, чтобы лучше понять их особенности в языке программирования Julia.

Целочисленные данные в Julia

В языке программирования Julia существует несколько типов данных для целых чисел, которые различаются по размеру и диапазону значений. Эти типы включают как знаковые (`Int8` , `Int16` , `Int32` , `Int64` , `Int128`), так и беззнаковые целые числа (`UInt8` , `UInt16` , `UInt32` , `UInt64` , `UInt128`). Каждый из этих типов имеет свои ограничения по диапазону значений,

которые могут быть представлены с использованием определённого количества бит.

Для наглядности и понимания, давайте выведем значения, которые могут быть представлены каждым из этих типов, для получения минимального и максимального значений используются функции typemin и typemax:

In [28]:
```
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: [-170141183460469231731687303715884105728,170141183460469231731687303
03715884105727]
  UInt8: [0,255]
 UInt16: [0,65535]
 UInt32: [0,4294967295]
 UInt64: [0,18446744073709551615]
UInt128: [0,340282366920938463463374607431768211455]
```

Пример взят из книги: **"Краткое описание языка программирования Julia и некоторые примеры его использования"** (Белов Глеб Витальевич).

В языке программирования Julia переменные по умолчанию имеют тип, аналогичный 64-битному целому числу со знаком (то есть тип Int64). Это значит, что когда вы пишете целое число без указания его типа, Julia автоматически интерпретирует его как Int64. Чтобы убедиться в этом, можно воспользоваться функцией **`typeof`**, которая возвращает тип данных для заданного значения.

In [16]:
```
typeof(1)
```

Out[16]:
```
Int64
```

Большие целые числа, которые не могут быть представлены с использованием 64 бит, но могут быть представлены в 128 битах будут иметь тип Int128.

In [21]:
```
typeof(134223234545412312321 3)
```

Out[21]:
```
Int128
```

Для явного указания типа, можно использовать синтаксис, который определяет размер числа:

In [34]:
```
typeof(Int8(1))
```

Out[34]:
```
Int8
```

Выбор подходящего типа данных для целых чисел имеет значение с точки зрения производительности и использования памяти. Например, если вы уверены, что значения чисел будут в пределах диапазона Int8, то использование этого типа сэкономит память, поскольку он занимает всего 1 байт.

Беззнаковые целые числа вводятся и выводятся с использованием префикса 0x. Префикс 0x используется для обозначения чисел в шестнадцатеричной системе счисления. Это позволяет записывать числа, используя цифры от 0 до 9 и буквы от a до f (или от A до F).

In [8]:
```
typeof(0x0a)
```

Out[8]:
```
UInt8
```

In [19]:
```
typeof(0x1233432456789abcdef)
```

Out[19]:
```
UInt128
```

Для двоичных чисел используется префикс 0b.

In [5]:
```
b = 0b1010 # Двоичное представление числа 10
```

Out[5]:
```
0x0a
```

Для восьмеричных чисел используется префикс 0o.

In [6]:
```
c = 0o12  # Восьмеричное представление числа 10
```

Out[6]:
```
0x0a
```

При выводе числа оно автоматически отображается в шестнадцатеричной системе как 0x0a

Значения, слишком большие для типов Int128, UInt128, при получают специальный тип BigInt. Размер типа BigInt зависит только от доступной оперативной памяти.

In [22]:
```
typeof(10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Out[22]:
```
BigInt
```

In [23]:
```
typeof(0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

Out[23]: BigInt

Вещественные данные в Julia

Как и с целыми числами, существуют числа с плавающей точкой разной длины в битах. По умолчанию числа с плавающей точкой в Julia имеют тип Float64.

In [41]:
```
typeof(0.24324)
```

Out[41]: Float64

In [24]:
```
typeof(0.)
```

Out[24]: Float64

In [25]:
```
typeof(.3)
```

Out[25]: Float64

Можно использовать экспоненциальную форму представления вещественного числа:

In [26]:
```
1e10
```

Out[26]: 1.0e10

In [27]:
```
typeof(1e10)
```

Out[27]: Float64

Для того чтобы использовать значения типа Float32, то необходимо использовать f вместо e:

In [29]:
```
typeof(0.5f0)
```

Out[29]: Float32

In [30]:
```
2.5f-4
```

Out[30]: 0.00025f0

In [42]:
```
typeof(Float16(0.32323))
```

Out[42]: Float16

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

In [32]:
```
2.0^1000
```

Out[32]:
```
1.0715086071862673e301
```

In [33]:
```
BigFloat(2.0)^1000
```

Out[33]:
```
1.071508607186267320948425049060001810561404811705533607443750388370351051124936e+301
```

BigFloat не назначается автоматически при вводе, а требует явного объявления для использования.

Комплексные чисела

In [2]:
```
3 + 4im
```

Out[2]:
```
3 + 4im
```

In [3]:
```
typeof(3 + 4im)
```

Out[3]:
```
Complex{Int64}
```

In [2]:
```
typeof(3.1 + 4im)
```

Out[2]:
```
ComplexF64 (alias for Complex{Float64})
```

1.3. Определение переменных в Julia

По умолчанию Julia автоматически определяет тип данных переменной в зависимости от присваиваемого значения. Однако в некоторых случаях, чтобы избежать ошибок или повысить производительность, можно явно указать тип данных для переменной. Система типов в Julia гибридная, то есть сочетает элементы как динамической, так и статической типизации.

Динамическая типизация

В Julia переменные не привязаны к определённому типу, и их типы могут изменяться на протяжении выполнения программы.

In [5]:
```
a = 2
typeof(a)
```

Out[5]:
```
Int64
```

In [6]:
```
a = 9.56
typeof(a)
```

Out[6]:
```
Float64
```

In [7]:
```
a = 998754345678906543567890876543567890
typeof(a)
```

Out[7]:
```
Int128
```

In [13]:
```
a = -2.0
typeof(a)
```

Out[13]:
```
Float64
```

In [9]:
```
sqrt(a)
```

```
DomainError with -2.0:
sqrt was called with a negative real argument but will only return a complex
result if called with a complex argument. Try sqrt(Complex(x)).

Stacktrace:
 [1] throw_complex_domainerror(f::Symbol, x::Float64)
   @ Base.Math ./math.jl:33
 [2] sqrt(x::Float64)
   @ Base.Math ./math.jl:686
 [3] top-level scope
   @ In[9]:1
```

Ошибка возникает, потому что Julia по умолчанию интерпретирует числовое значение как Float64, что является типом для вещественных чисел. При попытке работы с комплексными числами, если явно не указать тип данных, Julia может не правильно интерпретировать выражение, что приводит к ошибке.

Статическая типизация

Несмотря на динамическую основу, Julia предоставляет возможность явно задавать типы переменных. Это важная особенность, которая помогает компилятору оптимизировать выполнение программы и избежать ошибок.

In [66]:
```
k::Complex = -2.0
typeof(k)
```

Out[66]:
```
ComplexF64 (alias for Complex{Float64})
```

In [67]:
```
sqrt(k)
```

Out[67]:
```
0.0 + 1.4142135623730951im
```

In [68]:
```
b::Int64 = 10
typeof(k)
```

Out[68]:
```
ComplexF64 (alias for Complex{Float64})
```

Если переменной типа Int64 попытаться присвоить значение с плавающей точкой, Julia выдаст ошибку, предупреждая о несоответствии типов.

In [52]:
```
b = 3.14
```

```
InexactError: Int64(3.14)

Stacktrace:
 [1] Int64
   @ ./float.jl:912 [inlined]
 [2] convert(::Type{Int64}, x::Float64)
   @ Base ./number.jl:7
 [3] top-level scope
   @ In[52]:1
```

Проверка быстродействия

Отказ от динамической типизации в пользу статической может значительно улучшить производительность программ. Для того чтобы наглядно показать, как это влияет на быстродействие, можно провести тест с использованием функции @time, чтобы измерить время выполнения кода с явной типизацией и без неё.

In [56]:
```
g::UInt64 = 18446744073709551615
h::UInt64 = 18446744073709551615

@time for i in 1:1000000 # Выполняем действие 1 миллион раз
    g + h
end
```

```
 0.000001 seconds
```

In [57]:
```
k = 18446744073709551615
f = 18446744073709551615

@time for i in 1:1000000
    k + f
end
```

```
 0.014609 seconds (1000.00 k allocations: 30.518 MiB, 7.60% gc time)
```

В результате сравнения кода с явной статической типизацией и без неё становится очевидным, что статическая типизация значительно ускоряет выполнение программы.

1.3.1. Присваивание и привязывание значений к переменным

В Julia оператор = используется для присваивания значения переменной. Однако если быть точным, то, что Julia делает, является не присваиванием, а привязыванием. В целях более глубокого понимания механизма работы

рассмотрим пример кода, в котором переменная x сначала привязывается к значению 2. Затем она повторно привязывается к значению x + 3:

In [2]:
```
#Определяем целочисленную переменную
x = 2
println("x = ", x)
#Вычисляем адрес переменной a с помощью функции  objectid.
println("Адрес переменной x = ", objectid(x))
```

```
x = 2
Адрес переменной x = 13228483051340567920
```

In [3]:
```
#Определяем целочисленную переменную
x = x + 3
println("x = ", x)
#Вычисляем адрес переменной a с помощью функции  objectid.
println("Адрес переменной x = ", objectid(x))
```

```
x = 5
Адрес переменной x = 14624617963239389700
```

Если бы этот пример исходного кода был написан на таком языке,как C/C++, Fortran или Pascal, то для хранения переменной x система выделила бы ячейку памяти. При каждом присваивании нового значения переменной x хранящееся в этой ячейке памяти число будет изменяться. В случае с привязыванием все работает иначе. Каждое вычисление нужно трактовать как создание числа, которое помещается в другую ячейку памяти. Привязывание предусматривает перемещение самой метки x в новую ячейку памяти. Переменная перемещается в результат, а не результат перемещается в переменную. Рассмотрим еще несколько примеров:

In [10]:
```
#Определяем вещественную переменную
x = 10.3
println("x = ", x)
#Вычисляем адрес переменной a с помощью функции  objectid.
println("Адрес переменной x = ", objectid(x))
```

```
x = 10.3
Адрес переменной x = 6536302453166368355
```

In [11]:
```
#Переопределяем вещественную переменную
x = -142.354
println("x = ", x)
#Вычисляем адрес переменной a с помощью функции  objectid.
println("Адрес переменной x = ", objectid(x))
```

```
x = -142.354
Адрес переменной x = 3856324898867229695
```

In [6]:
```
#Определяем строку
x="Пример строки"
println("x = ", x)
```

```
#Вычисляем адрес переменной a с помощью функции objectid.
println("Адрес переменной x = ", objectid(x))
```

```
x = Пример строки
Адрес переменной x = 13029857696698101524
```

Julia работает так, что переменная получает значение, вычисленное справа от знака равенства. Присваивание также является выражением, что означает, что его результат можно использовать в других вычислениях. Это позволяет делать несколько присваиваний одновременно или использовать их в более сложных выражениях.

In [4]:
```
x = (y = 6 + 4) * 5
println("x = ", x)
println("y = ", y)
```

```
x = 50
y = 10
```

1.4. Ввод-вывод данных

`Print`, **`println`** - это универсальные функции, которые можно использовать для вывода текста на экран. Давайте рассмотрим несколько простых примеров, чтобы продемонстрировать механизм работы этих функций:

In [8]:
```
println("hello"); println("world")
```

```
hello
world
```

In [9]:
```
print("hello"); print("world")
```

```
helloworld
```

В языке программирования Julia символ обратного слэша \ используется для экранирования специальных символов, таких как \n и \t.

1. \n - символ перевода строки. При выводе строки с символом \n, текст будет перенесен на новую строку.
2. \t - символ табуляции. При выводе строки с символом \t, добавляется горизонтальный отступ, эквивалентный ширине табуляции.

Пример использования в Julia:

In [10]:
```
print("hello\n"); print("world\n")
```

```
hello
world
```

In [11]:
```
print("hello \t world")
```

```
hello    world
```

Приведенный выше код показывает, что `println` – это тот же самый `print` с добавленным в конце символом новой строки \n.

Для ввода значений через клавиатуру в Julia можно использовать функцию `readline()`, которая считывает строку, введенную пользователем. Затем вы можете преобразовать эту строку в нужный тип данных с помощью функции `parse()`, если это необходимо. Вот пример:

In [13]:
```
println("Введите целое число:")
n = readline()

println("Вы ввели число: ", n)
println(typeof(n))

n = parse(Float64, n)
println("Число преобразованное в Float64: ", n)
println(typeof(n))
```

```
Введите целое число:
Вы ввели число: 1244
String
Число преобразованное в Float64: 1244.0
Float64
```

Округление вещественных чисел.

В Julia можно задать количество цифр после запятой, которое нужно выводить для вещественных чисел, используя функцию `round()`.

In [14]:
```
x = 3.14159265
println("x = ", x)
round_x = round(x, digits=2)
println("Округляем до 2 знаков после запятой = ", round_x)

x = 5.46
println("x = ", x)
round_x = round(x, digits=1)
println("Округляем до 1 знака после запятой = ", round_x)
```

```
x = 3.14159265
Округляем до 2 знаков после запятой = 3.14
x = 5.46
Округляем до 1 знака после запятой = 5.5
```

Выравнивание с помощью функций lpad и rpad.

С помощью функций дополнения можно указывать, что строковый литерал всегда должен иметь заданную длину. Если введенный текст меньше, то он будет дополнен выбранным знаком. Если знак не указан, то по умолчанию используется пробел.

In [15]:
```
lpad("ABC", 6, '-') #Дополнение слева.
```

Out[15]:
```
"---ABC"
```

In [16]:
```
rpad("ABC", 6, '-')#Дополнение справа.
```

Out[16]:
```
"ABC---"
```

In [17]:
```
lpad("", 10, '*')rpad("ABC", 10, '*')rpad("ABC", 10, '*')
```

Out[17]:
```
"**********ABC*******ABC*******"
```

Использование греческих букв и символов Юникода в Julia для математических вычислений

Язык Julia отличается тем, что в нем активно использует греческие буквы, такие как π, θ, α и Δ. Это связано с тем, что в математике и науке часто используются греческие символы для обозначения переменных и констант в уравнениях. Когда такие формулы реализуются в коде на Julia, использование греческих букв делает их более похожими на математические уравнения, что упрощает их чтение и понимание. Это делает язык Julia удобным для работы с математическими вычислениями.

Ниже приводится сводка из нескольких популярных греческих букв и символов Юникода, которые можно использовать в своем коде.

Символ	Заполнение по Tab
π	\pi
θ	\theta
Δ	\Delta
e	\euler
√	\sqrt
φ	\varphi

In [59]:
```
println(√121)
println(π * 2)
println(ℯ * 10)
```

```
11.0
6.283185307179586
27.18281828459045
```

Как известно из программы средней школы, выражения 3 × x + 2 × y записывают как 3x + 2y. Julia позволяет писать умножение таким же образом. Экземпляры такой записи называются литеральными

коэффициентами как своего рода аббревиатура умножения числового литерала на константу или переменную:

In [60]:
```
x = 5
x = 2x
println(x)
x = 2(10 + 15)
println(x)
```

```
10
50
```

1.5. Основные операторы языка Julia над числовыми значениями

1.5.1. Базовые арифметические операции

В арифметических выражениях можно испрользовать следующие знаки операций:

- `+` - сложение;
- `-` - вычитание;
- `*` - умножение;
- `/` - деление;
- `÷` - деление нацело;
- `%` - остаток от деления;
- `^` - возведение в степень.

`"+."`, `"-."`, `"*."`, `"/."`, `"^."` - поэлементные базовые арифметические операции (для векторов и матриц);

In [2]:
```
# Пример простейших арифметических выражений
println("Введите целое число a")
a = parse(Int, readline())
println("Введите целое число b")
b = parse(Int, readline())

println("a*b=", a * b) # Умножение
println("a/b=", a / b) # Деление
println("a÷b=", a ÷ b) # Целочисленное деление
println("a%b=", a % b) # Остаток от деления
println("a-b=", a - b) # Вычитание
println("a^b=", a ^ b) # Возведение в степень
```

```
Введите целое число a
Введите целое число b
```

```
a*b=55
a/b=2.2
a÷b=2
a%b=1
a-b=6
a^b=161051
```

В Julia есть проблема **возведения отрицательного числа в дробную степень**.

In [27]:
```julia
a=-8
b=a^(1/3)
print("a=",a,"a^(1/3)=",b);
```

```
DomainError with -8.0:
Exponentiation yielding a complex result requires a complex argument.
Replace x^y with (x+0im)^y, Complex(x)^y, or similar.

Stacktrace:
 [1] throw_exp_domainerror(x::Float64)
   @ Base.Math ./math.jl:41
 [2] ^(x::Float64, y::Float64)
   @ Base.Math ./math.jl:1206
 [3] ^(x::Int64, y::Float64)
   @ Base ./promotion.jl:456
 [4] top-level scope
   @ In[27]:2
```

In [28]:
```julia
# Для корректного возведения отрицатедьного числа в степень надо использоват
println("Введите число a")
a = parse(Float64, readline())
if a>0
    b=a^(1/3)
else
    b=-(abs(a)^(1/3))
end
print("a=",a,"\na^(1/3)=",b)
```

```
Введите число a
a=-4.0
a^(1/3)=-1.5874010519681994
```

В Julia существует возможность ввода строки, которая является арифметическим выражением.

Функция `Meta.parse()` в Julia преобразует строку в форму, которая может быть интерпретирована и выполнена как код. Это позволяет динамически создавать и выполнять код на лету.

Функция `eval()` в Julia используется для выполнения выражений, представленных в виде кода, переданных ей в качестве аргумента. Она позволяет вычислять и выполнять код во время выполнения программы.

In [59]:
```
stroka = "45*9+334"
println(stroka)
result = eval(Meta.parse(stroka))
println("Результат вычисления: ",result)
```

```
45*9+334
Результат вычисления: 739
```

In [1]:
```
println("Введите строку для вычисления:")
stroka = readline()
result = eval(Meta.parse(stroka))
println("Результат вычисления: ",result)
```

```
Введите строку для вычисления:
Результат вычисления: 739
```

Можно использовать символ двоеточия (:) для определения выражения, вместо Meta.parse, как в примере выше:

In [61]:
```
stroka = :(45*9+334)
println(stroka)
result = eval(stroka)
println("Результат вычисления: ",result)
```

```
45 * 9 + 334
Результат вычисления: 739
```

С помощью оператора интерполяции **$** можно использовать вычисленные значения при конструировании выражений:

In [55]:
```
x = 5
y = :($x + 10)
```

Out[55]:
```
:(5 + 10)
```

In [56]:
```
eval(y)
```

Out[56]:
```
15
```

In [57]:
```
y = :(45643 + 4567654)
```

Out[57]:
```
:(45643 + 4567654)
```

В Julia, как и в других языках программирования, существует несколько специальных значений для представления неопределенных или бесконечных величин: Inf, -Inf и NaN.

1. **Inf (бесконечность)** Inf — это специальное значение, которое представляет положительную бесконечность. Его можно использовать в вычислениях, где результат выходит за пределы конечных чисел.

In [38]:
```
x = Inf
println(x)
```

```
y = 1 / 0
println(y)
```

```
Inf
Inf
```

Отрицательная бесконечностьть:

In [39]:
```
z = -Inf
println(z)
```

```
-Inf
```

NaN (Not a Number) NaN — это специальное значение, которое используется для представления неопределенных или недопустимых результатов вычислений, например, при делении нуля на ноль.

In [42]:
```
a = 0 / 0
println(a)
```

```
NaN
```

Операции с Inf и NaN

- Inf + 1 дает Inf.
- Inf - Inf или Inf / Inf даст NaN, поскольку результат таких операций неопределен.
- Inf * 0 также даст NaN.
- Любая операция с NaN (например, NaN + 1, NaN * 2) всегда возвращает NaN.

In [44]:
```
x = Inf
y = 1 / 0  # Положительная бесконечность
z = -Inf    # Отрицательная бесконечность

println(x + y)  # бесконечность + бесконечность
println(x * 0)  # бесконечность * 0
println(Inf / Inf)  # деление бесконечности на бесконечность
```

```
Inf
NaN
NaN
```

1.5.2. Двоичные (арифметические) операторы (битовые операторы)

В языке Julia можно различить унарные и бинарные двоичные операторы над целыми значениями.

Унарные операции включают в себя операцию инверсии (~), где целое число переводится в двоичное представление и каждый бит

инвертируется.

Бинарные операции включают в себя:

- **Двоичное И (&)**, где оба операнда переводятся в двоичную систему, и над ними выполняется операция побитного И.
- **Двоичное ИЛИ (|)**, где оба операнда переводятся в двоичную систему, и над ними выполняется операция побитного ИЛИ.
- **Двоичное исключающее ИЛИ (^)**, где оба операнда переводятся в двоичную систему, и над ними выполняется операция побитного исключающего ИЛИ.
- **Сдвиг влево (<<)**, где первый операнд переводится в двоичную систему счисления, а затем смещается влево на количество позиций, определяемых вторым операндом (k), что эквивалентно умножению на 2^k.
- **Сдвиг вправо (>>)**, где первый операнд переводится в двоичную систему счисления, а затем смещается вправо на количество позиций, определяемых вторым операндом (k), что эквивалентно делению нацело на 2^k.

In [30]:
```
# Унарные операции
# Инверсия (~) - целое число переводится в двоичное представление и побитно
a = 13
println(a, " ", bitstring(a), " ", bitstring(~a), " ", ~a)

# Бинарные операции
# Двоичное И (&), оба операнда переводятся в двоичную систему и над ними поб
a = 13
b = 23
c = a & b
println("a=", a, " b=", b, " a&b=", c)

# Двоичное ИЛИ (|), оба операнда переводятся в двоичную систему и над ними п
a = 13
b = 23
c = a | b
println("a=", a, " b=", b, " a|b=", c)

# Двоичное исключающее ИЛИ (^), оба операнда переводятся в двоичную систему
a = 13
b = 23
c = a ⊻ b
println("a=", a, " b=", b, " a^b=", c)

# Сдвиг влево <<, первый операнд переводится в двоичную систему счисления и
a = 23
println(a, " ", a << 1, " ", a << 2, " ", a << 3)

# Сдвиг вправо >>, первый операнд переводится в двоичную систему счисления и
```

```
a = 20
println(a, " ", a >> 1, " ", a >> 2, " ", a >> 3)
```

```
13 00000000000000000000000000000000000000000000000000000000000001101 11111111
11111111111111111111111111111111111111111111111111110010 -14
a=13 b=23 a&b=5
a=13 b=23 a|b=31
a=13 b=23 a^b=26
23 46 92 184
20 10 5 2
```

1.5.3. Операторы отношения

В Julia определены следующие операторы отношения:

- `>` — больше;
- `<` — меньше;
- `==` — проверка эквивалентности объектов;
- `===` — проверка идентичности объектов;
- `!=` — не равно;
- `!==` — не идентично;
- `>=` — больше или равно;
- `<=` — меньше или равно;

In [125…
```
3 == 3.0
```

Out[125…
```
true
```

In [124…
```
3 === 3.0
```

Out[124…
```
false
```

In [126…
```
3.0 === 3.0
```

Out[126…
```
true
```

Julia позволяет проверять операции двойного неравенства, принятые в математике, например:

In [31]:
```
println("x = ")
x = parse(Float64, readline())
print(-7<x<=3)
```

```
x =
true
```

1.5.4. Логические операторы

Логические операторы языка Julia: `||` (или) и `&&` (и).

In [32]:
```
x = 3
x < 4 || x > 10
```

Out[32]:
```
true
```

In [33]:
```
x > 4 && x < 10
```

Out[33]:
```
false
```

1.5.5. Операторы присваивания

In [34]:
```
# Примеры операторов присваивания
x=3
println("x = ",x)
y=z=0.2^1.7
println("y = ",y," z = ",z)
x=3
a=4
println("x = ",x," a = ",a)
x+=a
println("x += a = ",x)
x=3
a=4
println("x = ",x," a = ",a)
x-=a
println("x -= a = ",x)
x=3
a=4
println("x = ",x," a = ",a)
x*=a
println("x *= a = ",x)
x=22
a=5
println("x = ",x," a = ",a)
x/=a
println("x /= a = ",x)
x=22
a=5
println("x = ",x," a = ",a)
x÷=a
println("x ÷= a = ",x)
x=22
a=5
println("x = ",x," a = ",a)
x%=a
println("x %= a = ",x)
x=22
a=5
println("x = ",x," a = ",a)
x^=a
println("x ^= a = ",x)
```

```
x = 3
y = 0.06482626386771051 z = 0.06482626386771051
x = 3 a = 4
x += a = 7
x = 3 a = 4
x -= a = -1
x = 3 a = 4
x *= a = 12
x = 22 a = 5
x /= a = 4.4
x = 22 a = 5
x ÷= a = 4
x = 22 a = 5
x %= a = 2
x = 22 a = 5
x ^= a = 5153632
```

1.6. Встроенные функции

Рассмотрим некоторые встроеные функции Julia.

- **`rem(a, b)`** - аналог a % b;
- **`div(a, b)`** - аналог a ÷ b;
- **`floor(a)`** - округляет число *a* в меньшую сторону;
- **`ceil(a)`** - округляет число *a* в большую сторону;
- **`round(a)`** - округляет число *a* до ближайшего целого числа.
- **`abs(a)`** — абсолютное значение числа a.
- **`sqrt(a)`** — квадратный корень числа a.
- **`cbrt(a)`** — кубический корень числа a.
- **`log(a)`** - натуральный логарифм числа a.
- **`log2(a)`** — логарифм a по основанию 2.
- **`log10(a)`** — десятичный логарифм a.
- **`log(n,a)`** - логарифм числа a по основанию n.

Тригонометрические функции если x в радианах, то **`sin(x)`**, **`cos(x)`**, **`tan(x)`**, **`cot(x)`**, **`asin(x)`**, **`acos(x)`**, **`atan(x)`**, **`acot(x)`**, **`sec(x)`**.

если x в градусах, то **`sind(x)`**, **`cosd(x)`**, **`tand(x)`**, **`cotd(x)`**, **`asind(x)`**, **`acosd(x)`**, **`atand(x)`**, **`acotd(x)`**, **`secd(x)`**.

- **`rad2deg(a)`** — преобразовать угол a из радиан в градусы
- **`deg2rad(a)`** — преобразовать угол a из градусов в радианы.

гиперболические функции: **sinh(x)**, **`cosh(x)`**, **`tanh(x)`**, **`coth(x)`**.

Гипотенуза **`hypot(a,b)`** - гипотенуза a и b

Комбинаторные функции

`factorial(a)` — факториал числа a

Ниже приведены примеры использования.

a = -23 b = 3.456 c = 7875 println(floor(b)) println(ceil(b)) println(round(b)) println(abs(a)) println(sqrt(b)) println(cbrt(b)) println(log(c)) println(floor(b)) println(rad2deg(2π)) println(deg2rad(180))

Глава 2. Структуры данных

2.1. Строки в julia

Строка в Julia – это набор символов, заключенных между двойными кавычками " ". Символы вводятся в кодировке UTF-8. Строки могут содержать специальные символы, например, символ табуляции '\t' или символ перевода на новую строку '\n':

In [37]:
```
b = "строка 1\nстрока 2\n"
println(b)
```

```
строка 1
строка 2
```

Функция **`length()`** - возвращает число символов в строке.

In [39]:
```
st = "Конь"
length(st)
```

Out[39]: 4

Строку можно рассматривать как одномерный массив (вектор). Например, если строка s="abc", то s[2]=='b', а s[end]=='c'. Однако изменять элементы строки присваиванием нельзя, т. е. оператор s[3]='d' является ошибочным с точки зрения языка Julia. В этом случае используется функция **`replace()`**.

In [40]:
```
s = "abc"
s[1]
```

Out[40]: 'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)

In [41]:
```
replace(s,'c'=>'a')
```

Out[41]: "aba"

Еще один нюанс, строка и символ — существенно разные понятия языка Julia,поэтому равенство "A" == 'A' является ложным.

Проверить наличие символа s в строке st можно с использованием конструкции **`s in st`**, которая возвращает true или false

In [42]:
```
'b' in "abc"
```

Out[42]:
```
true
```

Проверка того, что ss входит в st осуществляется с использованием функции **`occursin(ss,st)`**.

In [43]:
```
occursin("cc","Россия")
```

Out[43]:
```
true
```

`findfirst(ss,st)` - найти первое вхождение подстроки или символа ss в строке st. Если ss - строка, то результатом является первый и последний индексы подстроки ss в строке st. Если ss – символ, результатом будет индекс, который соответствует номеру символа в строке. Если подстрока или символ не найдены, функция возвращает nothing.

In [46]:
```
findfirst("ia","julia")
```

Out[46]:
```
4:5
```

In [47]:
```
findfirst('u',"julia")
```

Out[47]:
```
2
```

`findlast(ss,st)` - найти последнее вхождение подстроки или символа ss в строке st. Если ss - строка, то результатом является первый и последний индексы подстроки ss в строке st. Если ss – символ, результатом будет индекс, который соответствует номеру символа в строке. Если подстрока или символ не найдены, функция возвращает nothing.

In [48]:
```
findlast('m',"comment")
```

Out[48]:
```
4
```

Объединение производится с использованием символа звездочка `*`.

In [49]:
```
c="Hello,"
d="world!"
e=c*d
```

Out[49]:
```
"Hello,world!"
```

`strip(st)` - удаляет пробелы вначале и в конце строки если строка состоит из пробелов, то strip(st) возвращает пустую строку.

`isempty(st)` – проверяет, есть ли символы в строке, если нет, возвращает true, иначе — false.

`split(st,'R')` - разобрать строку на элементы, если в качестве разделителя используется символ R.

In [50]:
```
st = "        Кубанский государственный университет        "
strip(st)
```

Out[50]:
```
"Кубанский государственный университет"
```

In [51]:
```
isempty(st)
```

Out[51]:
```
false
```

In [52]:
```
s = "a,bc,d"
split(s,',')
```

Out[52]:
```
3-element Vector{SubString{String}}:
 "a"
 "bc"
 "d"
```

`join(a,'R')` - преобразовать элементы массива a в строку, используя в качестве разделителя символ R. join -оператор, обратный split.

In [53]:
```
a=[1.0, 2.0, 3.0]
join(a,',')
```

Out[53]:
```
"1.0,2.0,3.0"
```

`uppercase(st)` - преобразовать строку в верхний регистр.

`lowercase(st)` - преобразовать строку в нижний регистр.

`titlecase(st)` – преобразовать в верхний регистр первый символ каждого слова строки.

`string(x)` – превратить число x в строку.

In [55]:
```
println(uppercase(st))
println(titlecase(st))
```

```
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кубанский Государственный Университет
```

В Julia используется такое понятие, как интерполяция. Смысл его заключается в следующем. Если есть строка st="123.456", или число

a=3.14, то их значения можно «внедрять» (интерполировать) в строку с использованием знака доллара $:

In [57]:
```
print("1 + 2 = $(1 + 2)")
```

```
1 + 2 = 3
```

2.2. Массивы в julia

2.2.1. Способы объявления массива

В языке Julia существует довольно много способов объявления массива. Рассмотрим некоторые из них.

Простейший способ объявления пустого одномерного массива: `Type[]`, где Type - тип данных:

In [59]:
```
a=String[]
```

Out[59]:
```
String[]
```

In [60]:
```
b=Float64[]
```

Out[60]:
```
Float64[]
```

In [61]:
```
a=Array{Float64,1}(undef,3)
#одномерный массив с тремя элементами типа Float64, элементы массива не опре
```

Out[61]:
```
3-element Vector{Float64}:
 4.2439915824e-314
 5.0e-324
 0.0
```

In [62]:
```
a=Array{Float64,2}(undef,3,5)
#двумерный массив 3*5 – три строки, пять столбцов типа Float64, элементы мас
```

Out[62]:
```
3×5 Matrix{Float64}:
 5.0e-324     8.0e-323  8.0e-323  3.81959e-313  2.5e-323
 1.4854e-313  8.0e-323  8.0e-323  0.0           0.0
 2.3342e-313  8.0e-323  8.0e-323  6.4e-323      1.0e-323
```

In [63]:
```
a=Vector{Float64}(undef,10)
#одномерный массив с десятью элементами типа Float64, элементы массива не ог
```

Out[63]:
```
10-element Vector{Float64}:
 6.9231707766666e-310
 6.9231707766808e-310
 6.92317077610493e-310
 6.92317077675196e-310
 6.9231707767662e-310
 6.92317077610493e-310
 6.9231707768231e-310
 6.92317077688e-310
 6.9231707769227e-310
 6.9231707769796e-310
```

In [64]:
```
a=Matrix{Int64}(undef,2,5)
#двумерный массив 2*5 – две строки, пять столбцов типа Int64, элементы масси
```

Out[64]:
```
2×5 Matrix{Int64}:
 7  19  24  28  31
 8  23  27  29  34
```

In [65]:
```
a=[1 2 3 4]
# вектор-строка (элементы через пробел).
#При таком объявлении Julia создает двумерный массив с одной строкой и n сто
```

Out[65]:
```
1×4 Matrix{Int64}:
 1  2  3  4
```

In [66]:
```
a=[1, 2, 3, 4]
# вектор-столбец (элементы через запятую или точку с запятой,
#но при определении можно использовать разделители одного типа)
```

Out[66]:
```
4-element Vector{Int64}:
 1
 2
 3
 4
```

In [67]:
```
a=[1 2; 3 4] #двумерная матрица
```

Out[67]:
```
2×2 Matrix{Int64}:
 1  2
 3  4
```

In [68]:
```
a=zeros(3) # одномерный массив с тремя элементами типа Float64, с нулевыми з
```

Out[68]:
```
3-element Vector{Float64}:
 0.0
 0.0
 0.0
```

In [69]:
```
a=zeros(3,4) # двумерный массив 3*4 типа Float64, с нулевыми значениями элем
```

Out[69]:
```
3×4 Matrix{Float64}:
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
```

In [70]:
```
a=ones(3) # одномерный массив с тремя элементами типа Float64, с единичными
```

Out[70]:
```
3-element Vector{Float64}:
 1.0
 1.0
 1.0
```

In [71]:
```
a=ones(3,4) # двумерный массив 3*4 типа Float64, с единичными значениями эле
```

Out[71]:
```
3×4 Matrix{Float64}:
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
```

In [72]:
```
a=Int.(zeros(3)) #одномерный массив с тремя элементами типа Int64, с нулевым
```

Out[72]:
```
3-element Vector{Int64}:
 0
 0
 0
```

In [73]:
```
a=zeros(3,4,5) #двумерный массив 3*4*5 , с нулевыми значениями элементов
```

Out[73]:
```
3×4×5 Array{Float64, 3}:
[:, :, 1] =
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0

[:, :, 2] =
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0

[:, :, 3] =
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0

[:, :, 4] =
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0

[:, :, 5] =
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
```

In [74]:
```
a=fill(5,3,3) # двумерный массив 3*3 типа Int64
```

Out[74]:
```
3×3 Matrix{Int64}:
 5  5  5
 5  5  5
 5  5  5
```

In [75]:
```
a=rand(5) # одномерный массив из 5 элементов, заполненный случайными числами
```

Out[75]:
```
5-element Vector{Float64}:
 0.8794306657535889
 0.09639413403817987
 0.07769428637907472
 0.2324819390274271
 0.33783446872027856
```

In [76]:
```
a=rand(1:5,5) # одномерный массив из 5 элементов, заполненный случайными цел
```

Out[76]:
```
5-element Vector{Int64}:
 3
 5
 3
 2
 5
```

In [77]:
```
a = 5
b = 2
c = 20
x=collect(a:b:c) # создать одномерный массив, первый элемент которого равен
```

Out[77]:
```
8-element Vector{Int64}:
  5
  7
  9
 11
 13
 15
 17
 19
```

2.2.2. Работа с элементами массива

- Добавить элемент v в конец массива a можно с использованием функции **`push!(a,v)`**:

Если после функции стоит "!", то это значит, что исходные данные, поступающие в функцию, будут изменяться.
Если после функции Не стоит "!", то данные будут неизменными.

In [80]:
```
a = [3, 56, 75, 3]
push!(a,23)
```

Out[80]:
```
5-element Vector{Int64}:
  3
 56
 75
  3
 23
```

- Добавить элемент v в начало массива a можно с использованием функции **`pushfirst!(a,v)`**:

In [81]:
```
pushfirst!(a,3455)
```

Out[81]:
```
6-element Vector{Int64}:
 3455
    3
   56
   75
    3
   23
```

- Для массива a типа Vector определена операция вставки элемента x в произвольную позицию n: **`insert!(a,n,x)`**.

In [82]:
```
insert!(a,3,10000)
```

Out[82]:
```
7-element Vector{Int64}:
  3455
     3
 10000
    56
    75
     3
    23
```

- Последний элемент массива a можно удалить так: **`pop!(a)`**.

In [83]:
```
pop!(a)
print(a)
```

```
[3455, 3, 10000, 56, 75, 3]
```

- Первый элемент массива a можно удалить так: **`popfirst!(a)`**.

In [84]:
```
popfirst!(a)
print(a)
```

```
[3, 10000, 56, 75, 3]
```

- Удалить элемент массива в позиции n можно так: **`deleteat!(a, n)`**.

In [85]:
```
deleteat!(a, 2)
```

Out[85]:
```
4-element Vector{Int64}:
    3
   56
   75
    3
```

- Поменять порядок элементов массива на обратный: **`a=a[end:-1:1]`**.

In [86]:
```
a=a[end:-1:1]
```

Out[86]:
```
4-element Vector{Int64}:
  3
 75
 56
  3
```

- Определить длину массива: **`length(a)`**.

In [87]:
```
length(a)
```

Out[87]:
```
4
```

- Найти максимальное число массива: **`maximum(a)`**.

In [88]:
```
maximum(a)
```

Out[88]:
```
75
```

- Найти минимальное число массива: **`minimum(a)`**.

In [90]:
```
minimum(a)
```

Out[90]:
```
3
```

2.2.3. Выборки

- Выбрать элементы вектора a с i_1 по i_3 с шагом i_2: **`a[i1:i2:i3]`**.

In [92]:
```
a = [3, 4, 5, 0]
```

Out[92]:
```
4-element Vector{Int64}:
 3
 4
 5
 0
```

In [93]:
```
a[2:1:3]
```

Out[93]:
```
2-element Vector{Int64}:
 4
 5
```

In [94]:
```
b = [3 4 6; 4 9 2; 4 1 10]
```

Out[94]:
```
3×3 Matrix{Int64}:
 3  4   6
 4  9   2
 4  1  10
```

In [95]:
```
b[2:3,2:3] # выбрать элементы матрицы из строк 2, 3 и столбцов 2, 3.
```

Out[95]:
```
2×2 Matrix{Int64}:
 9   2
 1  10
```

In [96]:
```
b[1:2,:] # выбрать строки 1 и 2. Двоеточие на месте одного из индексов означ
```

Out[96]:
```
2×3 Matrix{Int64}:
 3  4  6
 4  9  2
```

Применительно к элементам массива можно использовать операции с точкой

In [97]:
```
a.^2 # вычислить квадрат всех элементов массива
```

Out[97]:
```
4-element Vector{Int64}:
  9
 16
 25
  0
```

In [98]:
```
a = [2, 4, 6]
b = [4, 10, 12]
a.+b
```

Out[98]:
```
3-element Vector{Int64}:
  6
 14
 18
```

In [99]:
```
a.*b
```

Out[99]:
```
3-element Vector{Int64}:
  8
 40
 72
```

In [100…
```
log.(a) # вычислить натуральный логарифм всех элементов массива.
```

Out[100…
```
3-element Vector{Float64}:
 0.6931471805599453
 1.3862943611198906
 1.791759469228055
```

In [101…
```
2 .*a
```

Out[101…
```
3-element Vector{Int64}:
  4
  8
 12
```

2.2.4. Объединение двух массивов

Если массивы a и b имеют одинаковое число столбцов, то объединить их (по вертикали) можно командой **`vcat(a,b)`** или **`[a;b]`**.

In [103…
```
a = [2.34, 4.355, 6.87]
b = [4, 10, 12]
vcat(a,b)
```

Out[103…
```
6-element Vector{Float64}:
  2.34
  4.355
  6.87
  4.0
 10.0
 12.0
```

In [104…
```
[a;b]
```

Out[104…
```
6-element Vector{Float64}:
  2.34
  4.355
  6.87
  4.0
 10.0
 12.0
```

Если массивы a и b имеют одинаковое число строк, то объединить их (по горизонтали) можно командой **`hcat(a,b)`** или **`[a b]`**.

In [105…
```
hcat(a,b)
```

Out[105…
```
3×2 Matrix{Float64}:
 2.34    4.0
 4.355  10.0
 6.87   12.0
```

In [106…
```
[a b]
```

Out[106…
```
3×2 Matrix{Float64}:
 2.34    4.0
 4.355  10.0
 6.87   12.0
```

2.2.5. Изменение размерности массива.

При необходимости можно изменить размерность массива с использованием функции `reshape()`. Допустим, нужно преобразовать одномерный массив a=[1,2,3,4,5,6] в двумерный с двумя строками и тремя столбцами. Это можно сделать следующим образом:

In [108…
```
a=[1,2,3,4,5,6]
a=reshape(a,2,3)
```

Out[108…
```
2×3 Matrix{Int64}:
 1  3  5
 2  4  6
```

In [109…
```
a=reshape(a,6)
```

Out[109…
```
6-element Vector{Int64}:
 1
 2
 3
 4
 5
 6
```

2.3. Словари

Словари — это неупорядоченные коллекции пар "ключ-значение" в языке программирования Julia. Они позволяют эффективно хранить и извлекать данные по ключу. Ключи в словарях должны быть уникальными и неизменяемыми, а значения могут быть любыми и изменяемыми.

2.3.1. Создание словарей

Словари создаются с использованием конструктора `Dict()` или с помощью литерального синтаксиса.

Примеры создания словарей:

In [110…
```
# Пустой словарь
d = Dict()

# Словарь с элементами
a = Dict("apple" => 1.2, "banana" => 0.8, "cherry" => 2.5)
```

Out[110…
```
Dict{String, Float64} with 3 entries:
  "cherry" => 2.5
  "banana" => 0.8
  "apple"  => 1.2
```

2.3.2. Доступ к элементам словаря

Доступ к значениям словаря осуществляется с помощью ключей. Если ключ не существует в словаре, возникает ошибка KeyError.

Пример:

In [112…
```julia
a = Dict("milk" => 1.5, "bread" => 2.0)

# Доступ к значениям
b = a["milk"]  # 1.5
print(b)
# Попытка доступа к несуществующему ключу
c = a["butter"]  # KeyError
```

```
1.5
```

```
KeyError: key "butter" not found

Stacktrace:
 [1] getindex(h::Dict{String, Float64}, key::String)
   @ Base ./dict.jl:498
 [2] top-level scope
   @ In[112]:7
```

2.3.3. Добавление и изменение элементов

Для добавления новых пар "ключ-значение" и изменения существующих используется синтаксис с квадратными скобками.

Пример:

In [113…
```julia
# Создание словаря
a = Dict("name" => "Alice", "age" => 30)
println(a)
# Добавление новой пары
a["city"] = "New York"
println(a)
# Изменение значения существующего ключа
a["age"] = 31
println(a)
```

```
Dict{String, Any}("name" => "Alice", "age" => 30)
Dict{String, Any}("name" => "Alice", "city" => "New York", "age" => 30)
Dict{String, Any}("name" => "Alice", "city" => "New York", "age" => 31)
```

2.3.4. Удаление элементов

Элементы можно удалять с помощью функции delete!.

Пример:

In [114…
```julia
# Создание словаря
a = Dict("name" => "Alice", "age" => 30, "city" => "New York")
println(a)
```

```
# Удаление элемента
delete!(a, "city")
println(a)
```

```
Dict{String, Any}("name" => "Alice", "city" => "New York", "age" => 30)
Dict{String, Any}("name" => "Alice", "age" => 30)
```

2.3.5. Проверка наличия ключа

Для проверки наличия ключа в словаре используется функция haskey.

Пример:

In [115…
```
a = Dict("milk" => 1.5, "bread" => 2.0)

# Проверка наличия ключа
println(haskey(a, "milk"))  # true
println(haskey(a, "butter"))  # false
```

```
true
false
```

2.3.5. Объединение словарей

Для объединения словарей можно использовать функцию merge.

Пример:

In [116…
```
dict1 = Dict("a" => 1, "b" => 2)
dict2 = Dict("c" => 3, "d" => 4)

# Объединение словарей
dict3 = merge(dict1, dict2)
```

Out[116…
```
Dict{String, Int64} with 4 entries:
  "c" => 3
  "b" => 2
  "a" => 1
  "d" => 4
```

2.4. Кортежи

Кортежи (tuples) - это упорядоченные, неизменяемые коллекции элементов различных типов. Они играют важную роль в языке программирования Julia благодаря своей эффективности и широкому применению в различных областях, таких как возвращение нескольких значений из функций, создание сложных ключей для словарей и др. Основные свойства кортежей

- Неизменяемость: В отличие от массивов, кортежи неизменяемы. Это означает, что после создания кортежа его содержимое не может быть изменено.

- Разнородность: Элементы кортежа могут быть разных типов.
- Упорядоченность: Элементы кортежа имеют определенный порядок и доступны по индексу.

2.4.1. Создание кортежей

Кортежи создаются с использованием круглых скобок и запятых для разделения элементов. В Julia можно создавать кортежи, содержащие элементы разных типов.

Примеры создания кортежей:

In [117…
```
t = (1, "hello", 3.5)
```

Out[117…
```
(1, "hello", 3.5)
```

Кортежи с одним элементом создаются с запятой, чтобы избежать двусмысленности:

In [118…
```
t = (1,)
```

Out[118…
```
(1,)
```

2.4.2. Доступ к элементам кортежа

Доступ к элементам кортежа

Элементы кортежа можно получить с помощью индексации. Индексация в Julia начинается с 1, а не с 0, как в некоторых других языках программирования.

In [119…
```
t = (10, 20, 30, 40)

# Доступ к первому элементу
first_element = t[1]  # 10
println(first_element)
# Доступ к последнему элементу
last_element = t[end]  # 40
println(last_element)
```

```
10
40
```

2.4.3. Распаковка кортежей

Распаковка позволяет присвоить элементы кортежа переменным.

In [120…
```
coordinates = (3, 5)
```

```
# Распаковка кортежа
x, y = coordinates

println("x = $x, y = $y")  # x = 3, y = 5
```

```
x = 3, y = 5
```

2.4.4. Неизменяемость кортежей

Попытка изменить элемент кортежа приведет к ошибке:

In [121…
```
t = (1, 2, 3)
t[1] = 10  # Ошибка! Кортежи неизменяемы.
```

```
MethodError: no method matching setindex!(::Tuple{Int64, Int64, Int64}, ::In
t64, ::Int64)

Stacktrace:
 [1] top-level scope
   @ In[121]:2
```

2.4.5. Вложенные кортежи

Кортежи могут содержать другие кортежи, создавая вложенные структуры.

Пример:

In [122…
```
t = (1, (2, 3), (4, (5, 6)))

# Доступ к элементам вложенного кортежа
a = t[2]  # (2, 3)
println(a)
b = t[3][2][1]  # 5
println(b)
```

```
(2, 3)
5
```

2.4.6. Преобразование кортежей

Хотя кортежи неизменяемы, можно создать новый кортеж на основе существующего, добавив или изменив элементы.

Пример:

In [123…
```
t = (1, 2, 3)

# Добавление элемента
new_t = (t..., 4)  # (1, 2, 3, 4)
println(new_t)
# Изменение элемента (создание нового кортежа)
```

```
modified_t = (t[1], 42, t[3])  # (1, 42, 3)
println(modified_t)
```

```
(1, 2, 3, 4)
(1, 42, 3)
```

Глава 3. Управляющие конструкции языка Julia

3.1. Условные операторы

3.1.1. Условный (тройной) оператор

Условный (тройной) оператор вида `a ? b : c` (обязательны пробелы слева и справа от символов ? и :), a – условие, если оно верно, то выполняется b, иначе выполняется c.

In [127…
```
x = -1
x > 0 ? println("x>0") : println("x<=0")
```

```
x<=0
```

Условие **x > 0** проверяет, больше ли **x** нуля. Поскольку **x = -1**, условие **x > 0** ложно следовательно выполняется выражение после `:` — `println("x<=0")`.

3.1.2. Условный оператор if

В Julia простейшая форма условного оператора имеет вид:

```
if логическое_выражение
        операторы, выполняемые, когда логическое выражение истинно
end
```

В такой форме действия после двоеточия выполняются, если логическое выражение истинно. Если же оно ложно, программа ничего не делает и переходит к следующему оператору. Полная форма оператора **if**:

```
if логическое_выражение
        операторы, выполняемые, когда логическое выражение истинно
else
        операторы, выполняемые, когда логическое выражение ложно
end
```

Если нужно последовательно проверить несколько условий, используется расширенная форма с дополнительным оператором **elseif**:

```
if логическое_выражение
        операторы, выполняемые, когда логическое выражение истинно
```

```
elseif второе_логическое_выражение
        операторы, выполняемые, когда второе логическое выражение
истинно
elseif третье_логическое_выражение
        операторы, выполняемые, когда третье логическое выражение
истинно
...
else
        операторы, выполняемые, когда все логические выражения
ложны
end
```

Дополнительных условий и связанных с ними блоков **elseif** может быть сколько угодно. Если некоторое условие оказалось истинным, соответствующий блок кода выполняется, и дальнейшие условия не проверяются.

3.1.3. Примеры задач

Задача 1. Решить квадратное уравнение.

Версия 1.

In [1]:
```
println("a = ")
a = parse(Float64, readline())
println("b = ")
b = parse(Float64, readline())
println("c = ")
c = parse(Float64, readline())

d = b * b - 4 * a * c
x1 = (-b + sqrt(d)) / 2 / a
x2 = (-b - sqrt(d)) / 2 / a

print("x1 = ",x1,"\t","x2 = ",x2)
```

```
a =
b =
c =
x1 = -0.25      x2 = -1.0
```

Версия 2.

In [2]:
```
println("a = ")
a = parse(Float64, readline())
println("b = ")
b = parse(Float64, readline())
println("c = ")
c = parse(Float64, readline())

d = b * b - 4 * a * c

if d >= 0
```

```
    x1 = (-b + sqrt(d)) / 2 / a
    x2 = (-b - sqrt(d)) / 2 / a
    print("x1 = ",x1,"\t","x2 = ",x2)
else
    print("Действительных корней нет")
end
```

```
a =
b =
c =
Действительных корней нет
```

Версия 3. Исправим ошибку вычисления корня из отрицательного числа.

In [3]:
```
println("a = ")
a = parse(Float64, readline())
println("b = ")
b = parse(Float64, readline())
println("c = ")
c = parse(Float64, readline())

d = b * b - 4 * a * c

if d >= 0
    x1 = (-b + sqrt(d)) / 2 / a
    x2 = (-b - sqrt(d)) / 2 / a
    print("x1 = ",x1,"\t","x2 = ",x2)
else
    x1 = (-b + sqrt(Complex(d))) / 2 / a
    x2 = (-b - sqrt(Complex(d))) / 2 / a
end

print("x1 = ",x1,"\t","x2 = ",x2)
```

```
a =
b =
c =
x1 = -0.6 + 1.0198039027185568im        x2 = -0.6 - 1.0198039027185568im
```

Задача 2. Напишите программу решения биквадратного уравнения.

In [4]:
```
println("Введите коэффициенты для биквадратного уравнения ax^4 + bx^2 + c =
a = parse(Float64, readline())
b = parse(Float64, readline())
c = parse(Float64, readline())

d = b^2 - 4*a*c

if d >= 0
    y1 = (-b + sqrt(d)) / (2*a)
    y2 = (-b - sqrt(d)) / (2*a)
else
    y1 = (-b + sqrt(Complex(d))) / (2*a)
    y2 = (-b - sqrt(Complex(d))) / (2*a)
end
```

```
x11 = sqrt(y1)
x12 = -sqrt(y1)
x21 = sqrt(y2)
x22 = -sqrt(y2)
println("x11=", x11, " x12=", x12, " x21=", x21, " x22=", x22)
```

```
Введите коэффициенты для биквадратного уравнения ax^4 + bx^2 + c = 0
x11=0.540007387273509 + 0.9442499554196238im x12=-0.540007387273509 - 0.9442
499554196238im x21=0.540007387273509 - 0.9442499554196238im x22=-0.540007387
273509 + 0.9442499554196238im
```

Задача 3. Напишите программу решения кубического уравнения.

In [5]:
```
println("a = ")
a = parse(Float64, readline())
println("b = ")
b = parse(Float64, readline())
println("c = ")
c = parse(Float64, readline())
println("d = ")
d = parse(Float64, readline())

r = b / a
s = c / a
t = d / a

p = (3 * s - r^2) / 3
q = (2 * r^3) / 27 - (r * s) / 3 + t
D = (p / 3)^3 + (q / 2)^2

if d < 0
    P = sqrt(Complex(-(p^3)) / 27)
    F = -q / (2 * P)
    F = π / 2 - atan(F / sqrt(Complex(1 - F^2)))
    x1 = 2 * P^(1/3) * cos(F / 3) - r / 3
    x2 = (2 * P^(1/3)) * (cos(F / 3) + 2 * π / 3) - r / 3
    x3 = (2 * P^(1/3)) * (cos(F / 3) + 4 * π / 3) - r / 3
    println("x1 = ", x1)
    println("x2 = ", x2)
    println("x3 = ", x3)
else
    u = (-q / 2 + sqrt(Complex(D)))^(1/3)
    v = (-q / 2 - sqrt(Complex(D)))^(1/3)
    h = (-u + v) / 2 - r / 3
    g = sqrt(3) * (u - v) / 2
    x1 = u + v - r / 3
    println("x1 = ", x1)
end
```

```
a =
b =
c =
d =
x1 = 0.4128123039144102 - 0.8997900397939138im
```

3.2. Операторы цикла Julia

Существует два цикла в Julia:

- while
- for

3.2.1. Цикл while

Общая структура цикла **while**

```
while условие
        оператор 1
        …
        оператор n
end
```

Рассмотрим несколько примеров.

Программа ищет и выводит наибольшее отрицательное число из всех введенных пользователем значений, останавливаясь, когда вводится 0.

In [6]:
```
println("N = ")
N = parse(Float64, readline())
kp = 0
mx = 0
while N != 0
    if N < 0
        kp = kp + 1
        if kp == 1
            mx = N
        elseif N > mx
            mx = N
        end
    end
    println("N = ")
    N = parse(Float64, readline())
end
print(mx);
```

```
N =
N =
N =
N =
N =
-5.0
```

Операторы управления циклом. Оператор `continue` начинает следующий проход цикла, минуя оставшееся тело цикла (**for** или **while**). Оператор `break` досрочно прерывает цикл.

3.2.2. Оператор for

В Julia оператор цикла **for** позволяет пройтись по всем элементам любой последовательности (строки, списка, кортежа и т.д.)

Цикл можно организовать следующим образом i1 - первое значение, i2 - последнее, i3- шаг цикла, по умолчанию i3=1 и шаг можно не задавать.

```
for i in i1:i2
        оператор 1
        …
        оператор n
end
for i = i1:i2
        оператор 1
        …
        оператор n
end
for i in i1:i3:i2
        оператор 1
        …
        оператор n
end
for i = i1:i3:i2
        оператор 1
        …
        оператор n
end
```

Примеры задач.

Задача 1. Переменная x меняется от xn до xk с шагом dx. Значение y вычисляется по формуле $e^{sin(x)}cos(x)$

Найти сумму и произведение значений y, минимальное и максимальное значение y.

In [17]:
```
xn = parse(Float64, readline())
xk = parse(Float64, readline())
dx = parse(Float64, readline())
s = 0
p = 1
min = 3
max = -3
x = xn
for x in xn:dx:xk
    y = exp(sin(x)) * cos(x)
    println("x=$x \t y=$y")
    s += y
    p *= y
    if y < min
        min = y
    end
    if y > max
        max = y
```

```
    end
end
println("Сумма=$s\nПроизведение=$p\nМинимум=$min\nМаксимум=$max")
```

```
x=-5.0   y=0.7400430180559947
x=-4.6   y=-0.3029448561642916
x=-4.2   y=-1.1720543119921003
x=-3.8   y=-1.4584288987570933
x=-3.4   y=-1.2482912328577325
x=-3.0   y=-0.8596947254714136
x=-2.6   y=-0.5117349058686644
x=-2.2   y=-0.2621934057605559
x=-1.8   y=-0.08579770651361984
x=-1.4   y=0.06344385942618513
x=-1.0   y=0.23291133013811396
x=-0.6   y=0.46925561259253096
x=-0.2   y=0.8034791012023205
x=0.2    y=1.195464195103658
x=0.6    y=1.4516158335858316
x=1.0    y=1.253380767493447
x=1.4    y=0.45534477137668733
x=1.8    y=-0.6016570131129559
x=2.2    y=-1.3209087543836786
x=2.6    y=-1.4348412180400183
x=3.0    y=-1.1400385675133151
x=3.4    y=-0.7487825922121494
x=3.8    y=-0.42897526360339133
x=4.2    y=-0.20507212889574455
x=4.6    y=-0.0415197321954094
x=5.0    y=0.10872913262953789
Сумма=-5.049267691737826
Произведение=1.0098861615879018e-8
Минимум=-1.4584288987570933
Максимум=1.4516158335858316
```

3.3. Исключения

try/catch — это блок для обработки исключений

Пример 1: Ошибка выхода за пределы массива

In [49]:
```
# Массив с числами
arr = [1, 2, 3]

try
    # Попытка обратиться к несуществующему индексу массива
    println(arr[5])
catch e
    println("Ошибка: выход за пределы массива.")
end
```

```
Ошибка: выход за пределы массива.
```

Пример 2: Несоответctry # Попытка преобразования строки в число num = parse(Int, "abc") println(num) catch e println("Ошибка: не удалось преобразовать строку в число.") end твие типов (например, сложение строки и числа)

In [18]:
```
try
    # Попытка сложить строку и число
    result = "Hello" + 5
    println(result)
catch e
    println("Ошибка: несоответствие типов данных.")
end
```

```
Ошибка: несоответствие типов данных.
```

Пример 3: Ошибка с преобразованием типов

In [19]:
```
try
    # Попытка преобразования строки в число
    num = parse(Int, "abc")
    println(num)
catch e
    println("Ошибка: не удалось преобразовать строку в число.")
end
```

```
Ошибка: не удалось преобразовать строку в число.
```

Пример 4: Работа с файлами (например, файл не существует)

In [20]:
```
try
    # Попытка открыть несуществующий файл
    file = open("file.txt", "r")
    println(readline(file))
catch e
    println("Ошибка: файл не найден.")
end
```

```
Ошибка: файл не найден.
```

Подробнее о работе с файлами в Julia, включая создание, чтение и запись вы можете найти в **Главе 12**.

Глава 4. Функции в Julia

4.1. Базовый синтаксис определения функции.

Структура функции в языке Julia

```
function name(список параметров)
тело функции
```

```
end
```
name – имя функции (не обязательно), список параметров тоже не обязателен.Возвращаемой величиной является последнее значение или список значений (кортеж). Перед списком возвращаемых значений можно использовать ключевое слово return. Функция может ничего не возвращать, в этом случае тип возвращаемой величины Nothing.

Вызов функции

```
name(список параметров))
```
Ключевое слово return обеспечивает выход из функции и может встречаться в тексте функции несколько раз.

```
function test(n)
    if n < 0
        return "n < 0"
    elseif n==0
        return "n == 0"
    else
        return "n > 0"
    end
end
```
Рассмотрим на примере функции нахождения корней квадратного уравнения.

function kv(a,b,c) d=b*b-4*a*c if (d>=0) x1=(-b+sqrt(d))/2/a x2=(-b-sqrt(d))/2/a pr=1 else x1=0 x2=0 pr=0 end return (pr,x1,x2) end

println("a = ") a = parse(Float64, readline()) println("b = ") b = parse(Float64, readline()) println("c = ") c = parse(Float64, readline())

y=kv(a,b,c)

if (y[1]==1) print("x1=",y[2]," x2=",y[3],'\n'); else print("нет действительных корней") end

4.2. Однострочные функции.

В языке программирования Julia можно определять функции не только с использованием традиционного синтаксиса `function ... end`, но и с помощью более компактного синтаксиса, который позволяет определить функцию в одну строку. Такие функции называются **однострочными функциями**.

```
function_name(список параметров) = выражение
```

In [15]:
```
add(x, y) = x + y
```

```
println(add(5, 6))
```

```
11
```

4.3. Анонимные функции.

Анонимные функции в **Julia** — это функции, которые не имеют имени и определяются прямо в месте их использования.

Основной синтаксис анонимных функций:

Анонимные функции в **Julia** обычно создаются с использованием стрелочного синтаксиса `->`, который указывает на определение функции. Стрелка разделяет параметры функции от тела функции.

Пример:

In [12]:
```
r = x -> x^2
println(r(5))
```

```
25
```

Анонимные функции могут принимать несколько аргументов, и это делается аналогично обычным функциям.

In [16]:
```
r = (x, y) -> x + y
println(r(3, 4))
```

```
7
```

Анонимные функции могут возвращать несколько значений, если они заключены в кортеж (или массив). Это полезно, если нужно вернуть сразу несколько результатов.

In [17]:
```
r = (x, y) -> (x + y, x * y)
println(r(5, 6))
```

```
(11, 30)
```

По умолчанию анонимные функции в Julia предполагают, что тело функции состоит из одного выражения. Однако можно использовать более сложные блоки кода, если они обернуты в **`begin ... end`**. В таком случае анонимная функция может содержать несколько выражений.

In [18]:
```
r = x -> begin
    z = x^2
    z + 1
end
println(r(3))
```

```
10
```

In [19]:
```
arr = [1, 2, 3, 4, 5]
r = map(x -> x^2, arr) # map применяет анонимную функцию ко всем элементам м
println(r)
```

```
[1, 4, 9, 16, 25]
```

4.4. Блоки do

Блоки do создает анонимную функцию и передает ее в качестве первого аргумента внешней функции при вызове:

In [45]:
```
map([1, 2, 3]) do x
2x
end
```

Out[45]:
```
3-element Vector{Int64}:
 2
 4
 6
```

In [46]:
```
map([1, 2, 3], [1, 2, 3]) do x, y
x + y
end
```

Out[46]:
```
3-element Vector{Int64}:
 2
 4
 6
```

Глава 5. Примеры программ на языке Julia

ЗАДАЧА 8.1. Найти первые **M** чисел Армстронга. Число Армстронга – натуральное число, равно сумме своих цифр, возведённых в степень, равную количеству его цифр. Работает ли ваш код при M=32?

Мы решили проверить скорость работы двух программ, написанных на Python и Julia, которые находят и выводят первые M чисел Армстронга.

Код на Python:

```
import time

def armstrong(x):
    p = x
    x_str = str(x)
    l = len(x_str)
    sum_x = 0
    while x != 0:
```

```
        sum_x += (x % 10) ** l
        x = x // 10
    if sum_x == p:
        return True
    else:
        return False

def find_armstrong_numbers(M):
    m = 0
    x = 0
    while m < M:
        x += 1
        if armstrong(x):
            m += 1
            print(f"{m}) x={x}")
    print("Цикл завершился!")

print("Введите количество чисел Армстронга, которые хотите найти:
")
M = int(input())
start_time = time.time()
find_armstrong_numbers(M)
end_time = time.time()
print(f"Время выполнения: {end_time - start_time:.6f} секунд")
```

Результат работы программы

```
Введите количество чисел Армстронга, которые хотите найти: 20
1) x=1
2) x=2
3) x=3
4) x=4
5) x=5
6) x=6
7) x=7
8) x=8
9) x=9
10) x=153
11) x=370
12) x=371
13) x=407
14) x=1634
15) x=8208
16) x=9474
17) x=54748
18) x=92727
19) x=93084
20) x=548834
Цикл завершился!
Время выполнения: 0.823005 секунд
```

In [1]:

```
function armstrong(x)
    p = x
    x_str = string(x)
    l = length(x_str)
    sum_x = 0
    while x != 0
        sum_x += (x % 10)^l
        x = x ÷ 10
    end
    if sum_x == p
        return true
    else
        return false
    end
end

function find_armstrong_numbers(M)
    m = 0
    x = 0
    while m < M
        x += 1
        if armstrong(x)
            m += 1
            println("$m) x=$x")
        end
    end
    println("Цикл завершился!")
end

println("Введите количество чисел Армстронга, которые хотите найти: ")
M = parse(Int, readline())
start_time = time()
find_armstrong_numbers(M)
end_time = time()
println("Время выполнения: $(end_time - start_time) секунд")
```

```
Введите количество чисел Армстронга, которые хотите найти:
```

```
1) x=1
2) x=2
3) x=3
4) x=4
5) x=5
6) x=6
7) x=7
8) x=8
9) x=9
10) x=153
11) x=370
12) x=371
13) x=407
14) x=1634
15) x=8208
16) x=9474
17) x=54748
18) x=92727
19) x=93084
20) x=548834
21) x=1741725
22) x=4210818
23) x=9800817
24) x=9926315
25) x=24678050
26) x=24678051
27) x=88593477
28) x=146511208
Цикл завершился!
Время выполнения: 10.363824129104614 секунд
```

По результатам измерений времени выполнения программ на Python и Julia для поиска чисел Армстронга можно сделать следующие выводы:

Julia продемонстрировала значительно более высокую производительность по сравнению с Python. Время выполнения программы на Julia для нахождения и вывода первых M чисел Армстронга было заметно меньше, что свидетельствует о высокой эффективности и оптимизации языка. Это делает Julia предпочтительным выбором для задач, требующих высокой скорости работы и эффективного использования ресурсов.

ЗАДАЧА 8.2. Решить уравнения методом Ньютона-Рафсона

Метод Ньютона-Рафсона используется для нахождения корней нелинейного уравнения. Этот метод требует итеративного подхода для приближения к корню.

In [22]:
```julia
function newton_raphson(f, df, x0, tol=1e-7, max_iter=1000)
    x = x0
    for i in 1:max_iter
        fx = f(x)
```

```
        if abs(fx) < tol
            return x
        end
        dfx = df(x)
        if dfx == 0
            error("Производная равна нулю, метод не применим.")
        end
        x = x - fx / dfx
    end
    error("Максимальное количество итераций достигнуто, решение не найдено."
end

# Пример использования: f(x) = x^2 - 2, df(x) = 2x
f(x) = x^2 - 2
df(x) = 2x
x0 = 1.0

root = newton_raphson(f, df, x0)
println("Корень уравнения: ", root)
```

```
Корень уравнения: 1.4142135623746899
```

ЗАДАЧА 8.3. Вывести на экран первую тысячу чисел Хэмминга. Число Хэмминга - это положительное целое число вида $2^i \cdot 3^j \cdot 5^k$ для некоторых неотрицательных целых чисел i, j, и k. Первое число Хэмминга равно $1 = 2^0 \cdot 3^0 \cdot 5^0$, второе число Хэмминга равно $2 = 2^1 \cdot 3^0 \cdot 5^0$, третье число Хэмминга равно $3 = 2^0 \cdot 3^1 \cdot 5^0$, четвертое число Хэмминга равно $4 = 2^2 \cdot 3^0 \cdot 5^0$, пятое число Хэмминга равно $5 = 2^0 \cdot 3^0 \cdot 5^1$.

In [23]:
```
function heming(n)
    while n % 5 == 0
        n = n / 5
    end
    while n % 3 == 0
        n = n / 3
    end
    while n % 2 == 0
        n = n / 2
    end
    return n == 1
end

k = parse(Int64, readline())
count = 0
j = 1

while count < k
    if heming(j)
        count += 1
        println("№=", count, ",", j)
    end
    j += 1
end
```

№=1,1
№=2,2
№=3,3
№=4,4
№=5,5
№=6,6
№=7,8
№=8,9
№=9,10
№=10,12
№=11,15
№=12,16
№=13,18
№=14,20
№=15,24
№=16,25
№=17,27
№=18,30
№=19,32
№=20,36
№=21,40
№=22,45
№=23,48
№=24,50
№=25,54
№=26,60
№=27,64
№=28,72
№=29,75
№=30,80
№=31,81
№=32,90
№=33,96
№=34,100
№=35,108
№=36,120
№=37,125
№=38,128
№=39,135
№=40,144
№=41,150
№=42,160
№=43,162
№=44,180
№=45,192
№=46,200
№=47,216
№=48,225
№=49,240
№=50,243

ЗАДАЧА 8.4. Найти сумму квадратов первых n (100 ≤ n ≤ 1000) чисел, кратных 7.

In [24]:
```
function sum_squares(n)
    first = 7
    sum = 0
    for i in 0:(n-1)
        num = first + i * 7
        sum += num ^ 2
    end
    return sum
end

println("Введите количество чисел, для которых нужно найти сумму квадратов (
n = parse(Int, readline())
if 100 <= n <= 1000
    result = sum_squares(n)
    println("Сумма квадратов первых $n чисел, кратных 7: $result")
else
    println("Ошибка: n должно быть в диапазоне от 100 до 1000.")
end
```

```
Введите количество чисел, для которых нужно найти сумму квадратов (100 ≤ n ≤
1000):
Сумма квадратов первых 300 чисел, кратных 7: 443207450
```

Глава 6. Модули

Модули в Julia позволяют организовывать код в отдельные, независимые блоки, что упрощает управление большими проектами, улучшает структуру кода и способствует его повторному использованию.

6.1. Создание модулей

Модуль создаётся с помощью ключевого слова **`module`**, и завершается ключевым словом **`end`**.

In [1]:
```
module NewModule
# код модуля
function hello(name)
    println("Привет, $name !")
end
end
```

Out[1]:
```
Main.NewModule
```

В этом примере создаётся модуль с именем MyModule, в котором определена функция hello.

Чтобы использовать модуль, его нужно импортировать с помощью ключевого слова **`using`** или **`import`**.

In [2]:
```
# если модуль в том же файле
using .NewModule
NewModule.hello("Юля")
```

```
Привет, Юля !
```

В случае, если нужный нам файл находится не там же, где и исполняемый файл Julia, необходимо будет создать файл с модулем, с расширением `.jl`, а так же указать путь к файлу следующим образом: **`include("<путь к файлу><имя файла>.jl")`**

In [1]:
```
include("/home/jusya/Hello.jl")  # Подключаем модуль
using .Hello  # Используем модуль
Hello.hello() # Вызов функции внутри модуля
```

```
Привет мир!
```

Если вы хотите, чтобы какие-то функции или переменные из модуля были доступны напрямую, их нужно экспортировать с помощью ключевого слова **`export`**.

In [2]:
```
module Add
export add
function add(a, b)
    return a + b
end
end

using .Add
add(2, 3)
```

Out[2]:
```
5
```

Переменные, объявленные внутри модуля, не доступны напрямую извне, если они не экспортируются:

In [4]:
```
module Test1
export x1
x1 = 10
end

using .Test1
println(x1)
```

```
10
```

Если бы переменная не была экспортирована, доступ к ней был бы возможен только через полное имя модуля:

In [5]:
```
module Test2
x2 = 10
end
```

```
using .Test2
println(Test2.x2)
```

```
10
```

6.2. Модули внутри других модулей

В Julia можно создавать модули внутри других модулей:

In [22]:
```
module Test3
module Test4
    export hello
    function hello(name)
        println("Привет из Test4, ", name)
    end
end
end

using .Test3.Test4

hello("Алекс")
```

```
Привет из Test4, Алекс
```

In [23]:
```
module Test5

module Test6
    function hello(name)
        println("Привет из Test6, ", name)
    end
end

function hello(name)
    println("Привет из Test5, ", name)
end

end

using .Test5

Test5.hello("Алиса")
Test5.Test6.hello("Саша")
```

```
Привет из Test5, Алиса
Привет из Test6, Саша
```

6.3. Разрешение конфликтов имени

Если два модуля экспортируют одну и ту же функцию или переменную, можно использовать полное имя модуля для явного указания, какую именно функцию вы хотите использовать.

In [24]:
```
module A
    export f
    function f()
        println("Функция из модуля A")
    end
end

module B
    export f
    function f()
        println("Функция из модуля B")
    end
end

using .A
using .B

# Обращение через полное имя
A.f()  # Выведет: Функция из модуля A
B.f()  # Выведет: Функция из модуля B
```

```
Функция из модуля A
Функция из модуля B
```

Так же можно переименовать функцию при импорте с помощью ключевого слова `as`. Это позволяет избежать путаницы и легко различать функции с одинаковыми именами.

In [17]:
```
using .A
using .B

# Переименование функции из модуля A
using .A: f as fA

# Переименование функции из модуля B
using .B: f as fB

# Теперь можно использовать переименованные функции
fA()
fB()
```

```
Функция из модуля A
Функция из модуля B
```

6.4. Стандартные и пустые модули

В Julia есть **стандартные модули**, которые автоматически доступны, и **пустые модули**, которые можно создавать самостоятельно.

Стандартные модули, которые доступны всегда и не требуют загрузки:

1. `Core` — содержит все функциональные возможности, встроенные в язык.

2. `Base` — содержит базовый функционал.
3. `Main` — это модуль, в котором выполняется код, когда вы запускаете программу или работаете в REPL.

Модули автоматически содержат using Core, using Base. Если эти определения по умолчанию не нужны, модули можно определить с помощью ключевого слова `baremodule`, однако Core все еще будет импортироваться. Такие модули имеют следующую структуру:

```
baremodule ModuleName

end
```

Если вы не хотите использовать даже Core, то структура такого модуля будет выглядеть следующим образом:

```
ModuleName =  Module(:ModuleName, false, false)
```

Мы создадим модуль, в котором с помощью макроса `@eval` (подробное описание макросов можно найти в главе 7) определим функции для выполнения базовых арифметических операций: сложения, вычитания, умножения и деления.

In [23]:
```
# Создаём модуль без использования Core и Base
NoCoreArithmetic = Module(:NoCoreArithmetic, false, false)

@eval NoCoreArithmetic begin
    # Функция для сложения двух чисел
    add(x, y) = $(+)(x, y)

    # Функция для вычитания
    subtract(x, y) = $(-)(x, y)

    # Функция для умножения
    multiply(x, y) = $(*)(x, y)

    # Функция для деления
    divide(x, y) = $(/)(x, y)
end
```

Out[23]:
```
divide (generic function with 1 method)
```

In [24]:
```
# Подключаем модуль
using .NoCoreArithmetic

# Используем функции, определённые в модуле
println("Сложение: ", NoCoreArithmetic.add(10, 5))
println("Вычитание: ", NoCoreArithmetic.subtract(10, 5))
println("Умножение: ", NoCoreArithmetic.multiply(10, 5))
println("Деление: ", NoCoreArithmetic.divide(10, 5))
```

```
Сложение: 15
Вычитание: 5
Умножение: 50
Деление: 2.0
```

Глава 7. Макросы

Макросы в Julia похожи на функции, но есть несколько отличий. Они определяются с помощью ключевого слова `macro` (вместо `function`) и вызываются с символом `@` перед именем макроса. В отличие от функций, для вызова макроса не обязательно использовать скобки — достаточно указать параметры через пробел.

Пример:

```
@some_macro arg1 arg2
```

Макросы выполняются **до того, как код начнёт выполняться**. То есть, они изменяют сам код, а не его результаты. Когда вы вызываете макрос, он преобразует код в абстрактное синтаксическое дерево (AST), и это позволяет манипулировать выражениями и структурой кода.

Важно, что макросы работают не с конкретными значениями переменных, а с **выражениями** (то есть, с самими кусками кода). Это позволяет создавать программы, которые могут изменять или генерировать код во время работы. Это называется **метапрограммированием**.

7.1. Встроенные макросы

1. `@time`

Макрос `@time` измеряет время выполнения кода и сообщает о затраченном времени, а также о расходе памяти.

In [9]:
```
@time begin
    x = 0
    for i in 1:1000000
        x += i
    end
end
```

```
 0.028977 seconds (2.00 M allocations: 30.509 MiB)
```

2. `@elapsed`

Макрос **@elapsed** — это упрощённая версия **@time**. Возвращает время выполнения выражения, но не выводит его в консоль.

In [10]:
```
t = @elapsed begin
    x = 0
    for i in 1:1000000
        x += i
    end
end
println("Время выполнения: ", t)
```

```
Время выполнения: 0.028571126
```

3. **`@which`**

Макрос **@which** используется для того, чтобы узнать, какой метод будет вызван для конкретного выражения.

In [11]:
```
@which println("Hello, world!")
```

Out[11]: println(xs...) in Base at coreio.jl:4

4. **`@show`**

Макрос **@show** используется для вывода значений выражений на экран. Это полезно для отладки, чтобы видеть результаты вычислений прямо в процессе выполнения.

In [12]:
```
x = 42
@show x
```

```
x = 42
```

Out[12]: 42

5. **`@assert`**

Макрос **@assert** используется для проверки условий. Если условие ложно, макрос генерирует ошибку и выводит проблемное выражение.

In [13]:
```
x = 5
@assert x > 0   # Условие истинно, ошибок не будет

@assert x < 0   # Ошибка
```

```
AssertionError: x < 0

Stacktrace:
 [1] top-level scope
   @ In[13]:4
```

6. **`@debug`**

Макрос **@debug** используется для вывода отладочной информации, но его поведение зависит от переменной окружения **JULIA_DEBUG**. Если эта переменная не установлена или пуста, макрос будет игнорироваться. Если переменная установлена, например, в значение `"all"`, макрос выведет указанную строку вместе с дополнительной информацией (имя файла и номер строки, где был вызван макрос).

In [26]:
```
@debug "Это сообщение для отладки"
```

```
┌ Debug: Это сообщение для отладки
└ @ Main In[26]:1
```

7.2. Создание макросов

Теперь, когда мы рассмотрели основные встроенные макросы, давайте перейдем к созданию собственных макросов.

Синтаксис для создания макроса выглядит так:

```
macro имя_макроса(параметры)
    тело_макроса
end
```

In [15]:
```
macro hello(name)
    return :("Привет, " * $name)
end
```

Out[15]:
```
@hello (macro with 1 method)
```

In [19]:
```
@hello("Мир")
```

Out[19]:
```
"Привет, Мир"
```

Возврат кода. Макросы возвращают код в виде выражений, используя символ : (так называемое символьное выражение).

Вставка значений с помощью $. Для вставки значений переменных в код макроса используется символ `$`. Это важно, потому что макрос генерирует новый код, и для вставки значений нужно явно указать, что это не просто текст, а значение переменной.

In [20]:
```
macro add(a, b)
    println("Вычисляем сумму: ", a, " + ", b)
    return :( $a + $b )
end
```

Out[20]:
```
@add (macro with 1 method)
```

In [21]:
```
@add 3 5
```

```
Вычисляем сумму: 3 + 5
```

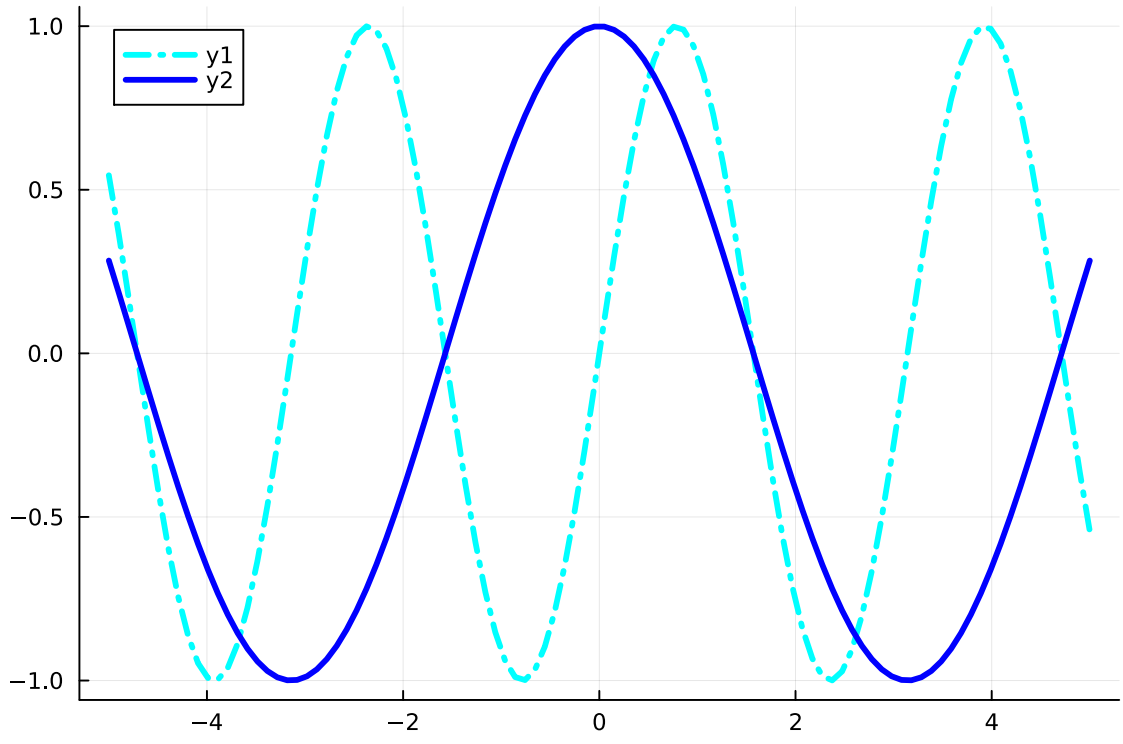
Out[21]: 8

Глава 8. Инструменты графического представления результатов

В Julia для визуализации данных часто используют пакет Plots. Этот пакет предоставляет единый интерфейс для множества различных бэкендов визуализации, таких как PyPlot (для работы с Matplotlib), GR (для работы с Gnuplot), PGFPlotsX и других. Вот несколько простых примеров использования пакета Plots для создания различных типов графиков.

In [25]:
```
using Plots
x = range(-5, stop = 5, length = 100)
y = sin.(2 .* x)
z = cos.(x)
plot(x, y, color=:cyan, width=3, linestyle = :dashdot)
plot!(x, z, color=:blue, linestyle =:solid, width=3)
```

Out[25]:



Для построения графика функция plot имеет вид `plot(x,y,[type])` x – массив абсцисс, y – массив ординат, параметр type является строкой, в которой определяется цвет графика и тип линии, в строку type можно включать следующие параметры

- **стиль линии** – строка, определяющая стиль линии

Атрибут	Стиль линии
dashdot	Штрихпунктирная линия
dash	Штриховая линия
dot	Пунктирная линия
solid	Жирная линия

- цвет – строка, определяющая цвет линии.

Атрибут	Цвет
red	Красный
green	Зелёный
blue	Синий
cyan	Голубой
purple	Пурпурный
yellow	Жёлтый
black	Чёрный
white	Белый

- **width** – строка, определяющая толщину линии.

Кроме просмотра графиков на экране современные графопостроители обязаны поддерживать вывод изображения в файл. В Plots это делается очень просто с помощью команд.

Сохранить график в файл в формате .png

```
savefig("myplot.png")
```
Или так

```
png("myplot.png")
```
Сохранить график в файл в формате .pdf

```
savefig("myplot.pdf")
```
Или так

```
pdf("myplot.png")
```
Если нужно построить на одном графике несколько функций, это можно сделать с использованием вызова функций с восклицательным знаком.

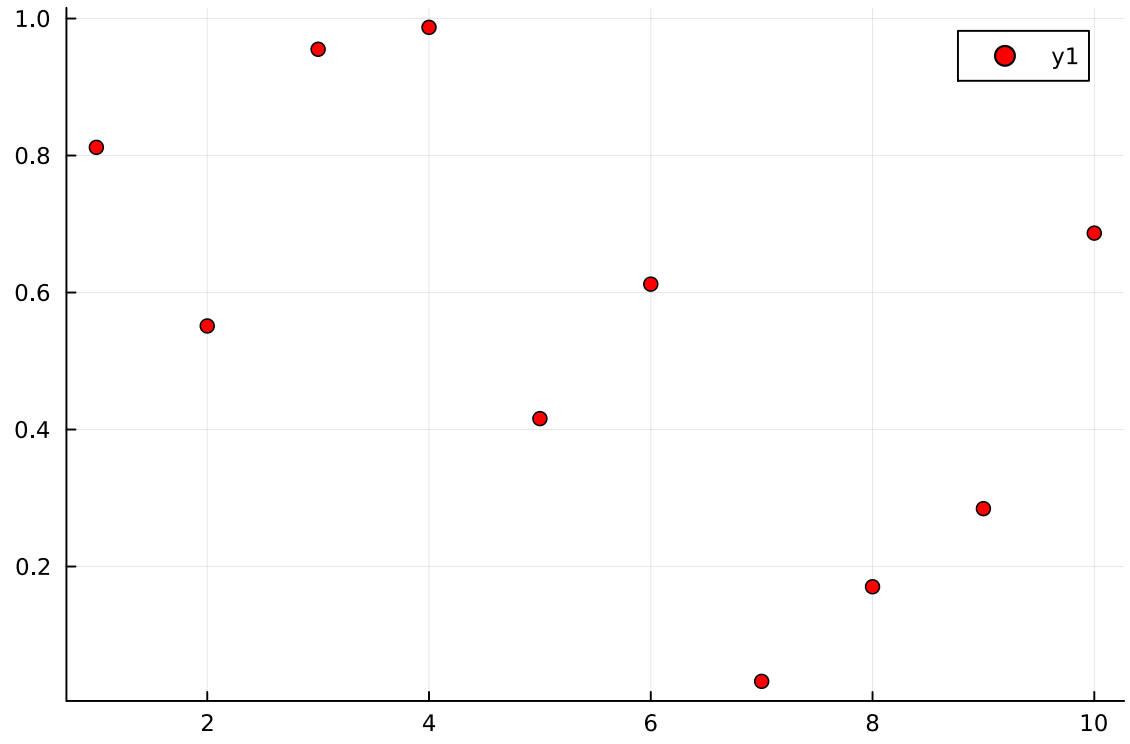
Функция `scatter`, используемая в библиотеке Plots, позволяет отображаеть данные в виде точек на плоскости.

```
scatter(x, y; [type])
```

x – массив абсцисс, y – массив ординат, type – дополнительные параметры (например, цвет точек, размер, маркеры и т. д.).

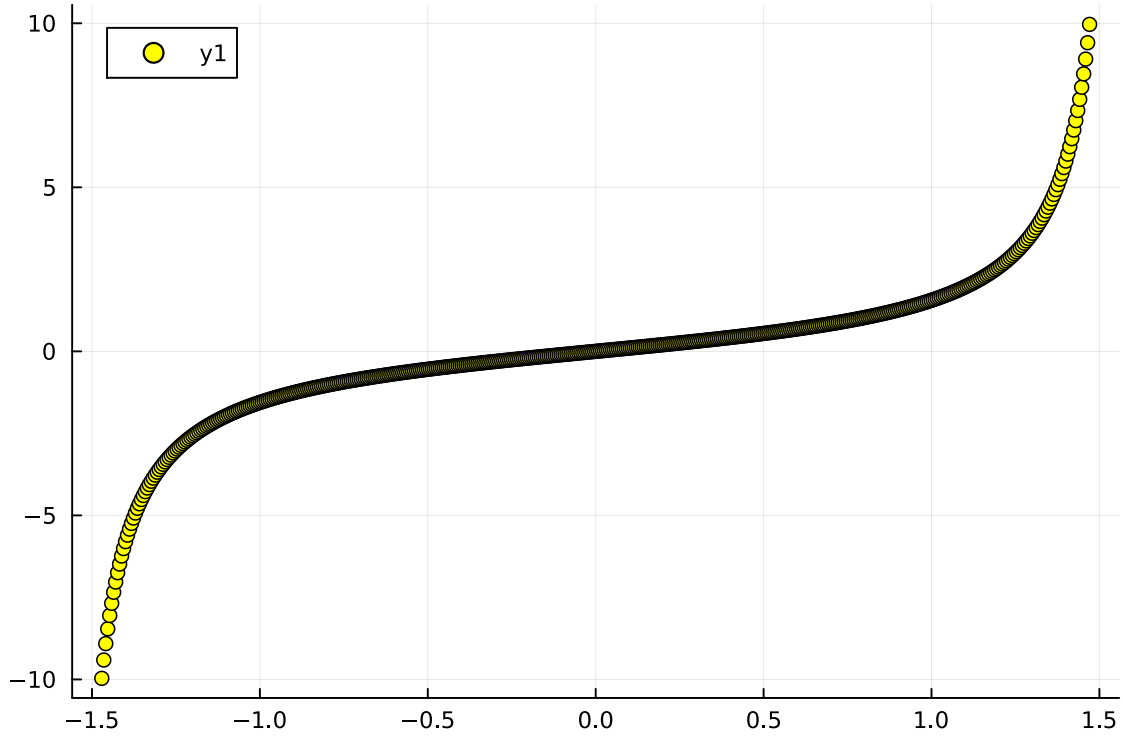
In [4]:
```
using Plots

x = 1:10
y = rand(10)

scatter(x, y, color=:red)
```

Out[4]:



In [32]:
```
using Plots

# Определяем диапазон значений для x
x = LinRange(-π/2 + 0.1, π/2 - 0.1, 500)


y = tan.(x)

scatter(x, y, color=:yellow)
```

Out[32]:

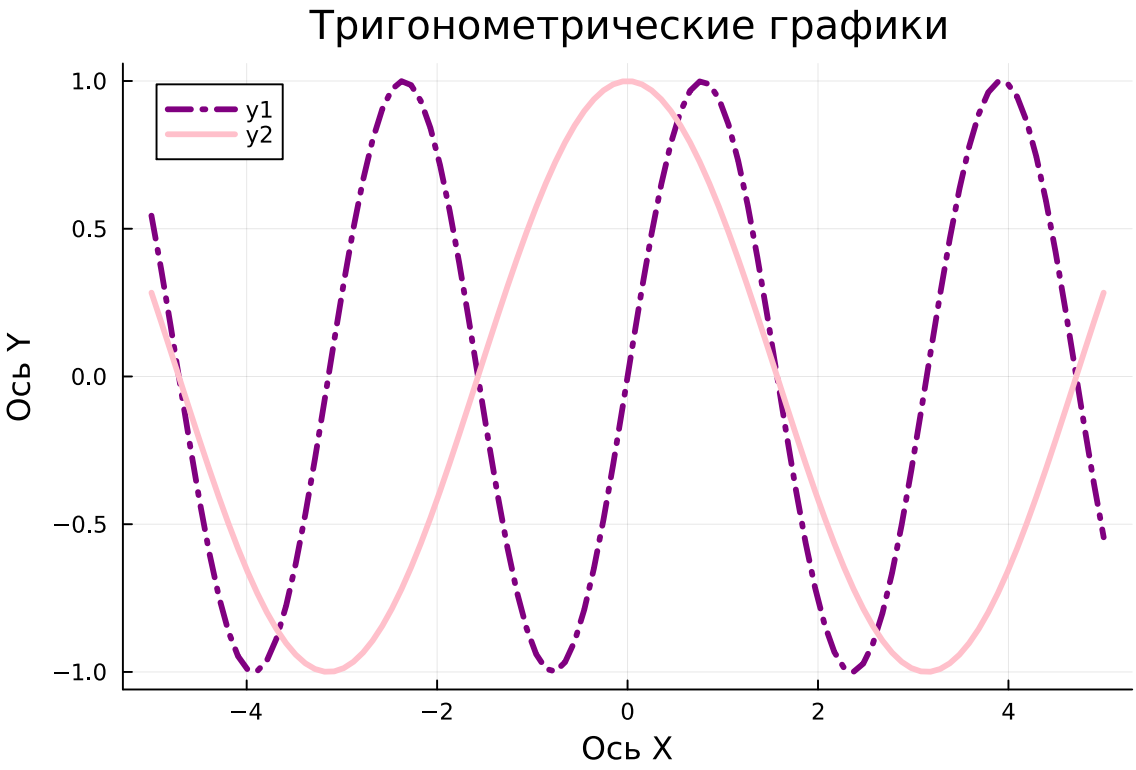


8.1. Заголовок, подписи, сетка, легенда

Простейшие подписи: заголовок и подписи к осям ставятся командами **`title`**, **`xlabel`** и **`ylabel`**, принимающими единственный аргумент – строку.

In [26]:
```
x = range(-5, stop = 5, length = 100)
y = sin.(2 .* x)
z = cos.(x)
plot(x, y, color=:purple, width=3, linestyle = :dashdot, title = "Тригономет
plot!(x, z, color=:pink, linestyle =:solid, width=3)
```

Out[26]:



Часто необходимо, чтобы в подписях на осях или легенде содержались верхние или нижние индексы, греческие буквы, различные значки и прочие математические символы. Для этого Plots имеет режим совместимости с командами LaTeX.В строке внутри пары символов ’ `\$` ’ можно вводить обычные формулы LaTeX ($ – начало и конец формулы)

Подпись графиков (вывод легенд):

- непосредственный вывод легенды осуществить командой **legend()**, . Чтобы переместить легенду или изменить её расположение, можно использовать: | Параметры | Строка | |:---------- |:----------| | best | наилучший | | topright | сверху справа| | topleft | сверху слева | | bottomleft | снизу слева| | bottomright | снизу справа| | left | по центру слева | | right | по центру справа | | bottom | снизу по центру | | top | сверху по центру| | inside | по центру |

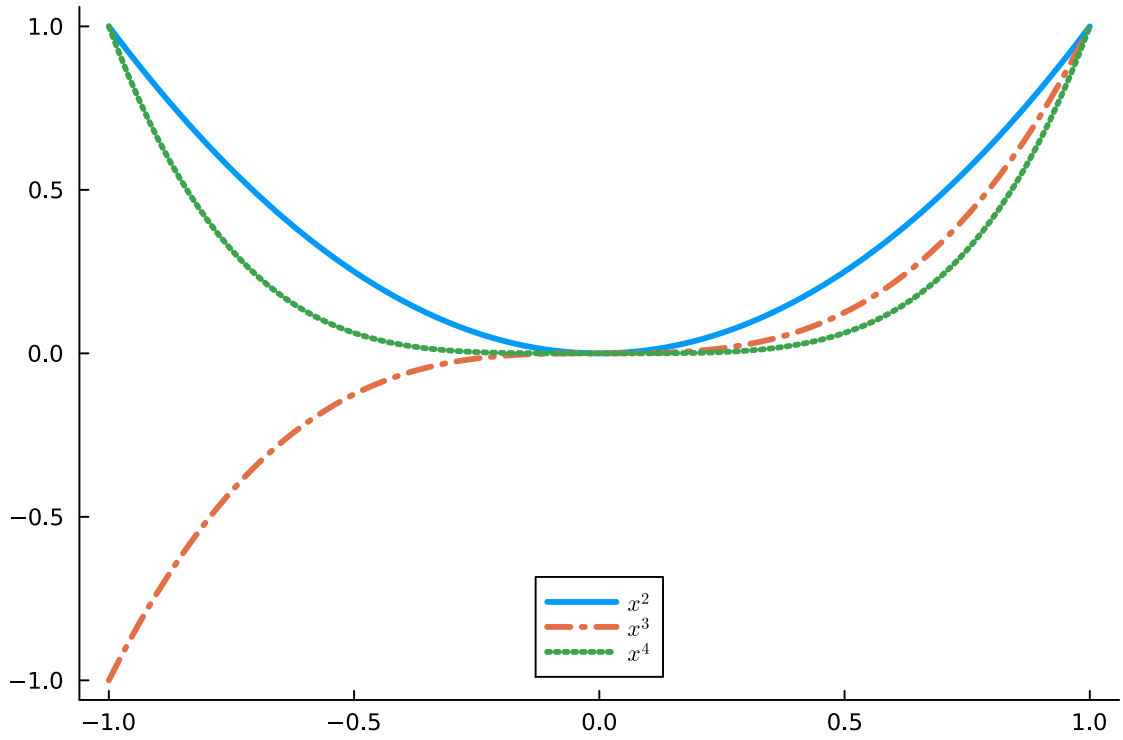
Чтобы отключить легенду, можно использовать **`legend=false`**.

- Ещё один часто встречающийся элемент графиков – сетка. Для получения сетки служит функция **grid** , у которой единственный логический аргумент (True – сетка будет, False – нет).

В качестве примера использования всего выше изложенного, рассмотри следующий пример.

In [28]:
```
t = range(-1, stop = 1, length = 100)
x = t .^2
y = t .^3
z = t .^4
plot(t , x , label = "\$x^2\$", legend=:bottom, grid=:false, width=3)
plot!(t , y , label = "\$x ^3\$",  linestyle = :dashdot, width=3)
plot!(t , z , label = "\$x ^4\$",  linestyle = :dot, width=3)
```

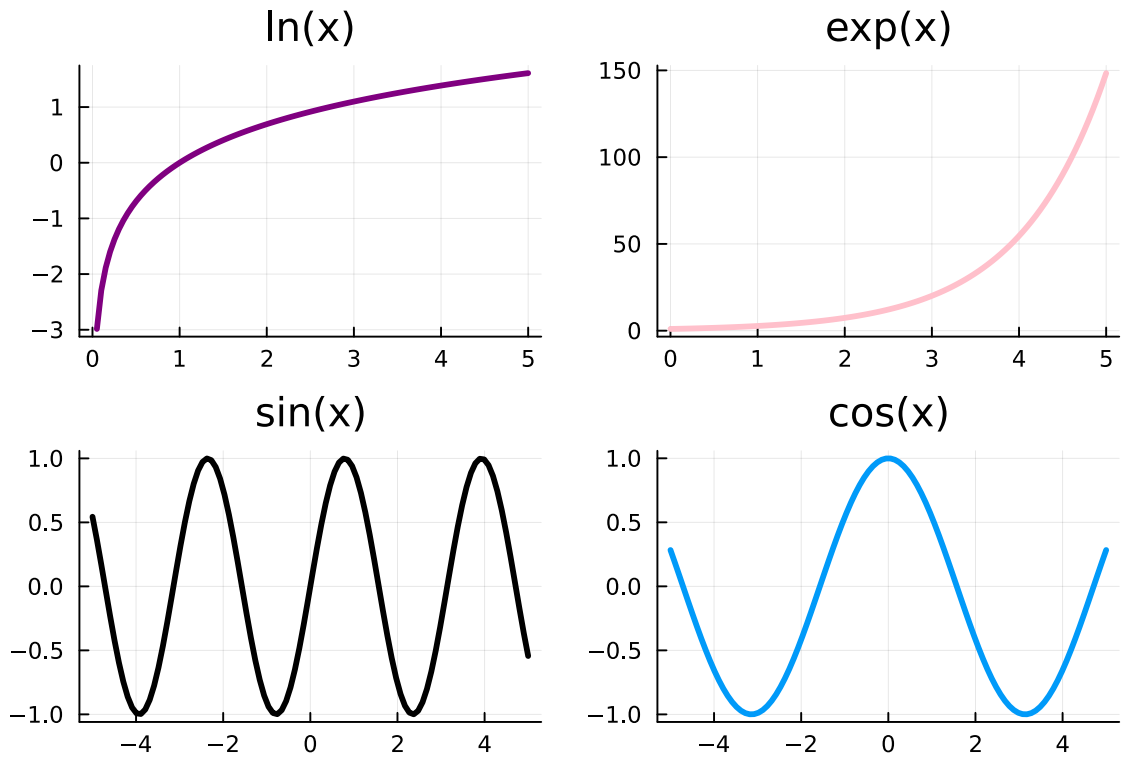
Out[28]:



8.2. Несколько графиков на одном полотне

In [30]:
```
x = range(0, stop = 5, length = 100)
y=log.(x)
p1 = plot(x,y, title="ln(x)", color=:purple, width=3, legend=false)
x = range(0, stop = 5, length = 100)
y=exp.(x)
p2 = plot(x,y, title="exp(x)", color=:pink, width=3, legend=false)
x = range(-5, stop = 5, length = 100)
y = sin.(2 .* x)
p3 = plot(x,y, title="sin(x)",  color=:black, width=3, legend=false)
x = range(-5, stop = 5, length = 100)
y = cos.(x)
p4 = plot(x,y, title="cos(x)", width=3, legend=false)
plot(p1, p2, p3, p4)
```

Out[30]:



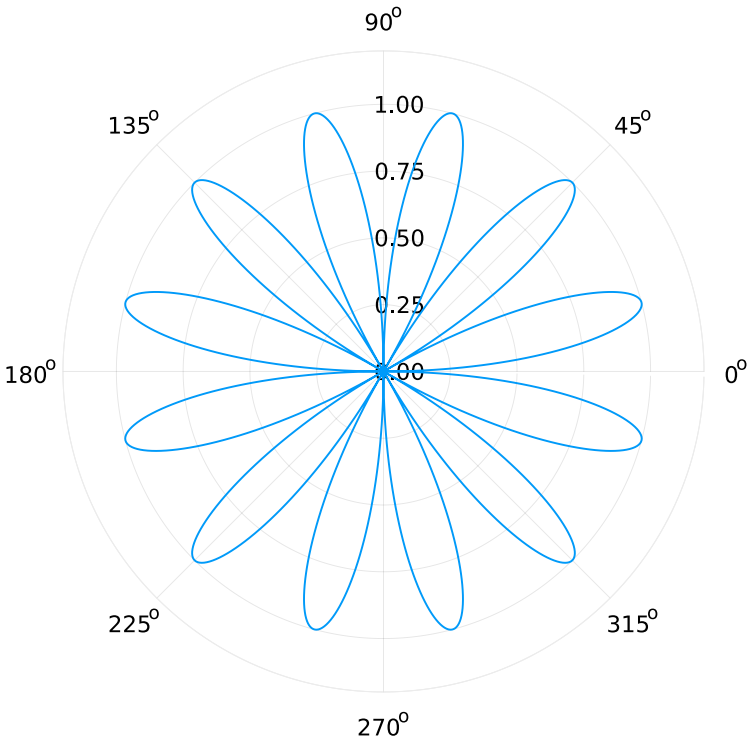
8.3. Графики в полярных координатах

Для построения графиков в полярных координатах нужно при вызове функции plot() указать proj=:polar.

Пример. Построить график следующей зависимости: r(t)=sin(6t), t∈[0;2π] в полярных координатах.

In [5]:
```
t=range(0,2pi, 1000)
rt=sin.(6.0.*t)
plot(t,rt,proj=:polar, legend=false)
```

Out[5]:



8.4. Трехмерные графики в Plots

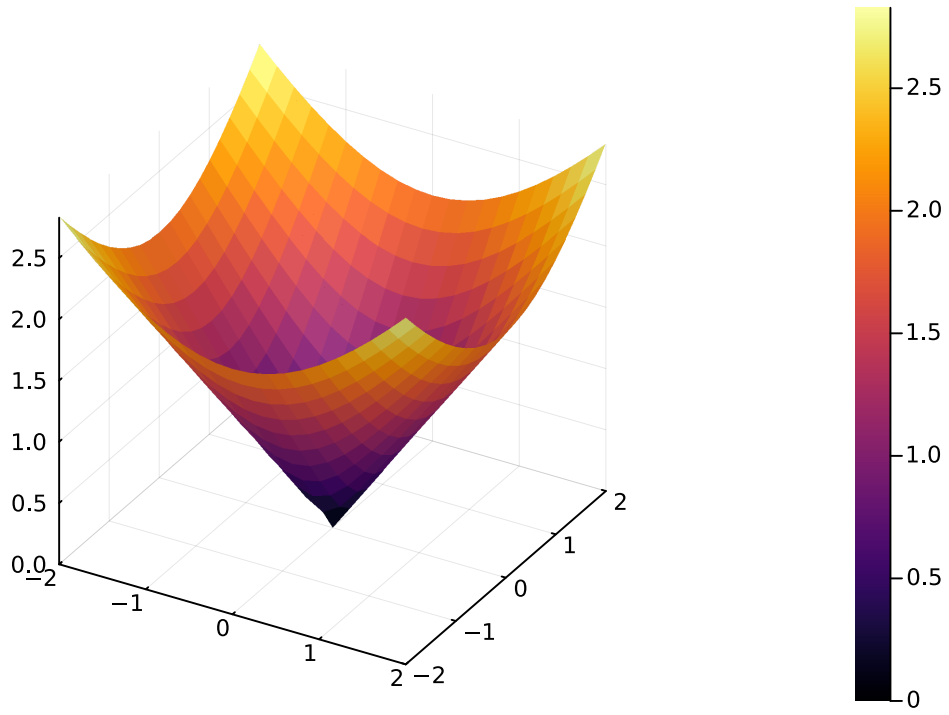
Для построения трехмерных графиков в Julia с использованием библиотеки Plots можно использовать функцию surface, которую необходимо использовать внутри функции **`plot`**.

```
plot(surface(x, y, z))
```

Задача 1. Построить график функции $z(x, y) = \sqrt{x^2 + y^2}$

In [4]:
```julia
using Plots

# Создаем сетку значений x и y
x = -2:0.2:2
y = -2:0.2:2

# Создаем матрицы X, Y для сетки
X = [xi for xi in x, yi in y]
Y = [yi for xi in x, yi in y]

# Вычисляем значения Z = sqrt(x^2 + y^2)
Z = [sqrt(xi^2 + yi^2) for (xi, yi) in zip(X, Y)]#функция zip позволяет пара
# Строим 3D график
plot(surface(X, Y, Z))
```

Out[4]:

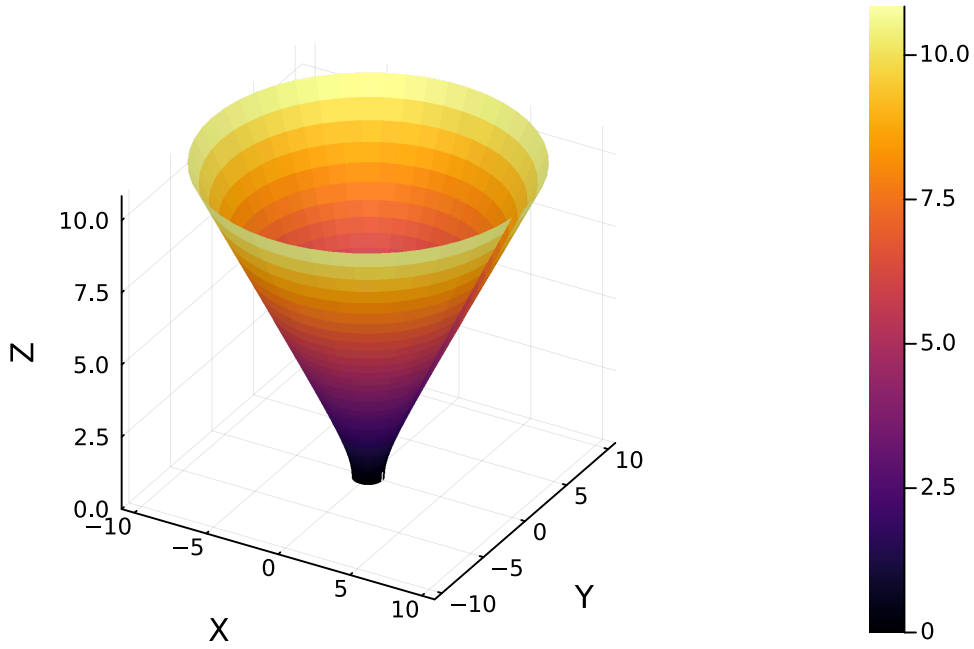


Задача 2. Построить поверхность однополостного гиперболоида, уравнение которого задано в параметрическом виде: $x(uv) = ch(u)cos(v)$, $y(u,v) = ch(u)sin(v)$, $z(u,v) = sh(u)$.

In [1]:

```
using Plots

# Шаг для параметров
h = π / 50

# Формируем вектор u и v
u = 0:h:π
v = 0:2h:2π


x = [cosh(ui) * cos(vi) for ui in u, vi in v]
y = [cosh(ui) * sin(vi) for ui in u, vi in v]
z = [sinh(ui) for ui in u, vi in v]

# Строим 3D-график поверхности
plot(surface(x, y, z), xlabel="X", ylabel="Y", zlabel="Z", title="ГРАФИК ОДН
```

Out[1]:

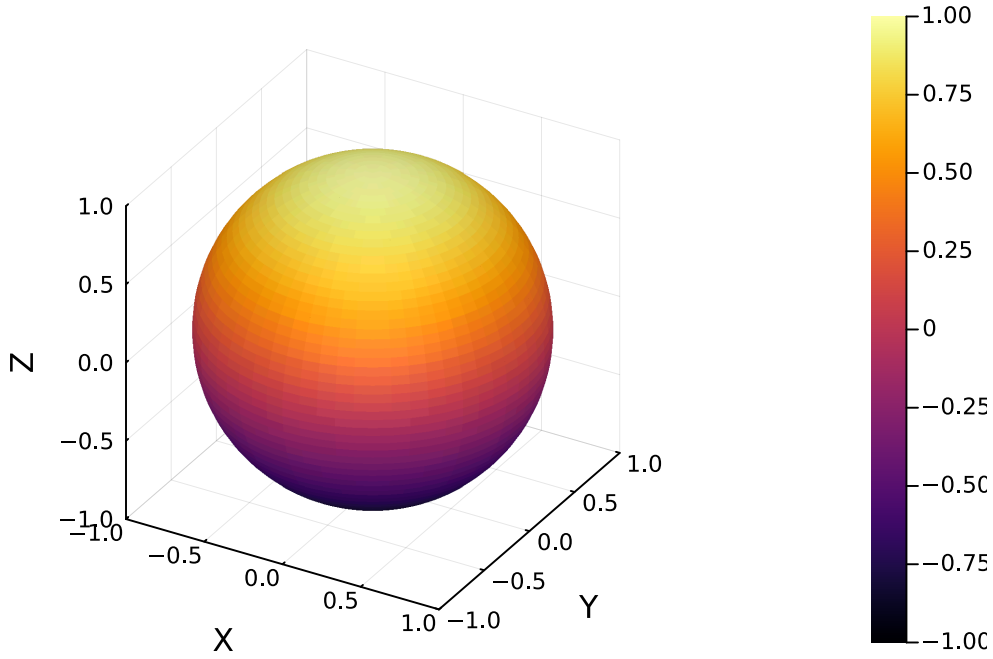


Задача 3. Построить поверхность сферы, уравнение которой задано в параметрическом виде: $x(u,v) = sin(u)cos(v)$, $y(u,v) = sin(u)sin(v)$, $z(u,v) = cos(u)$.

In [5]:

```julia
using Plots

# Шаг для параметров
h = π / 60

# Формируем вектор u и v
u = 0:h:π
v = 0:2h:2π

x = [sin(ui) * cos(vi) for ui in u, vi in v]
y = [sin(ui) * sin(vi) for ui in u, vi in v]
z = [cos(ui) for ui in u, vi in v]

# Строим 3D-график поверхности
plot(surface(x, y, z), xlabel="X", ylabel="Y", zlabel="Z", title="СФЕРА")
```

Out[5]:



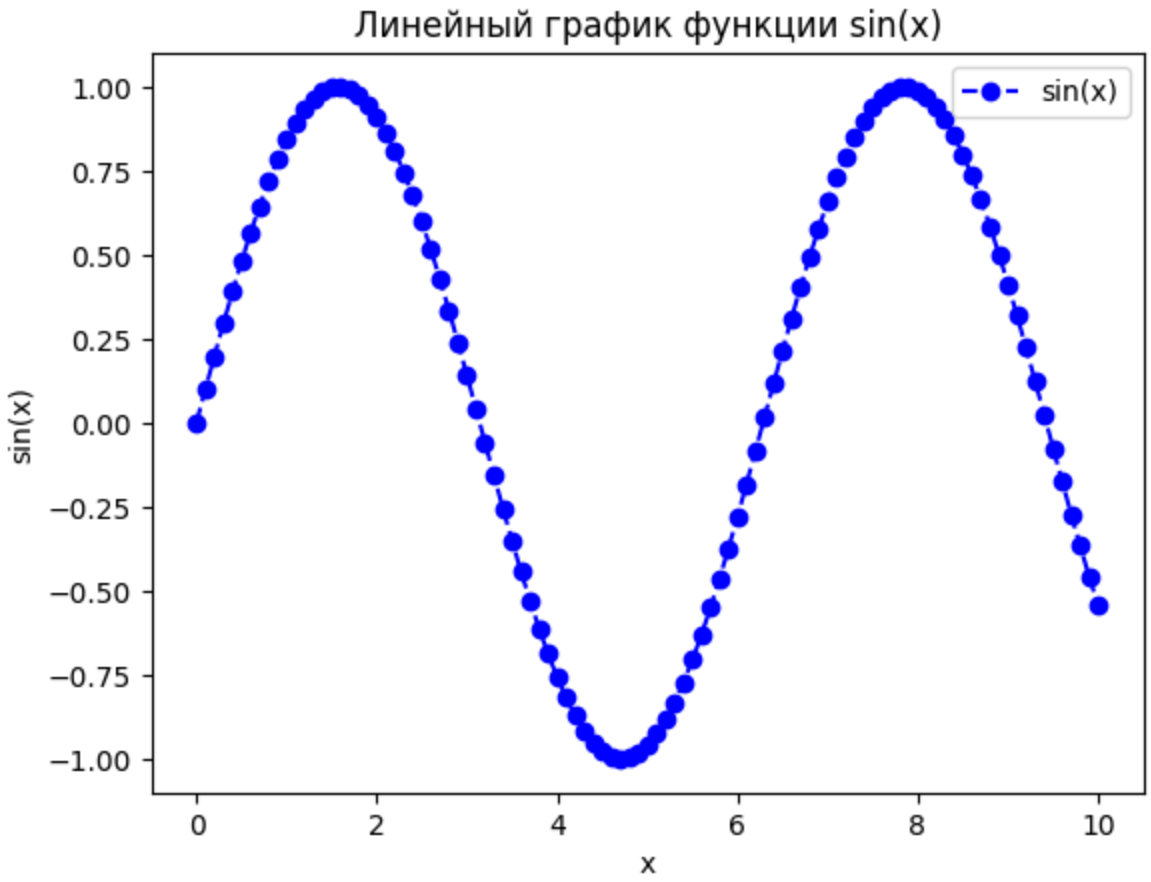
8.5. Библиотека PyPlot и ее использование с Plots

Для визуализации данных в Julia также существует библиотека PyPlot, которая предоставляет интерфейс для работы с популярной Python-библиотекой matplotlib.

Вот пример использования библиотеки PyPlot в Julia:

In [2]:
```julia
using PyPlot

#График с одной линией
x = collect(0:0.1:10)  # Преобразуем StepRange в массив
y = sin.(x)  # Применяем sin к каждому элементу массива x

figure()  # Создаем новый график
plot(x, y, label="sin(x)", color="blue", linestyle="--", marker="o")
xlabel("x")  # Подписываем ось X
ylabel("sin(x)")  # Подписываем ось Y
title("Линейный график функции sin(x)")  # Заголовок графика
legend()  # Легенда
show()  # Показываем график

#График с несколькими линиями
y2 = cos.(x)  # Применяем cos к каждому элементу x

figure()  # Создаем новый график
plot(x, y, label="sin(x)", color="blue")
plot(x, y2, label="cos(x)", color="red")
```

```
xlabel("x")
ylabel("y")
title("График sin(x) и cos(x)")
legend()
show()

#Гистограмма
data = randn(1000)  # Генерация случайных данных

figure()  # Создаем новый график
hist(data, bins=30, edgecolor="black")
xlabel("Значения")
ylabel("Частота")
title("Гистограмма случайных данных")
show()
```



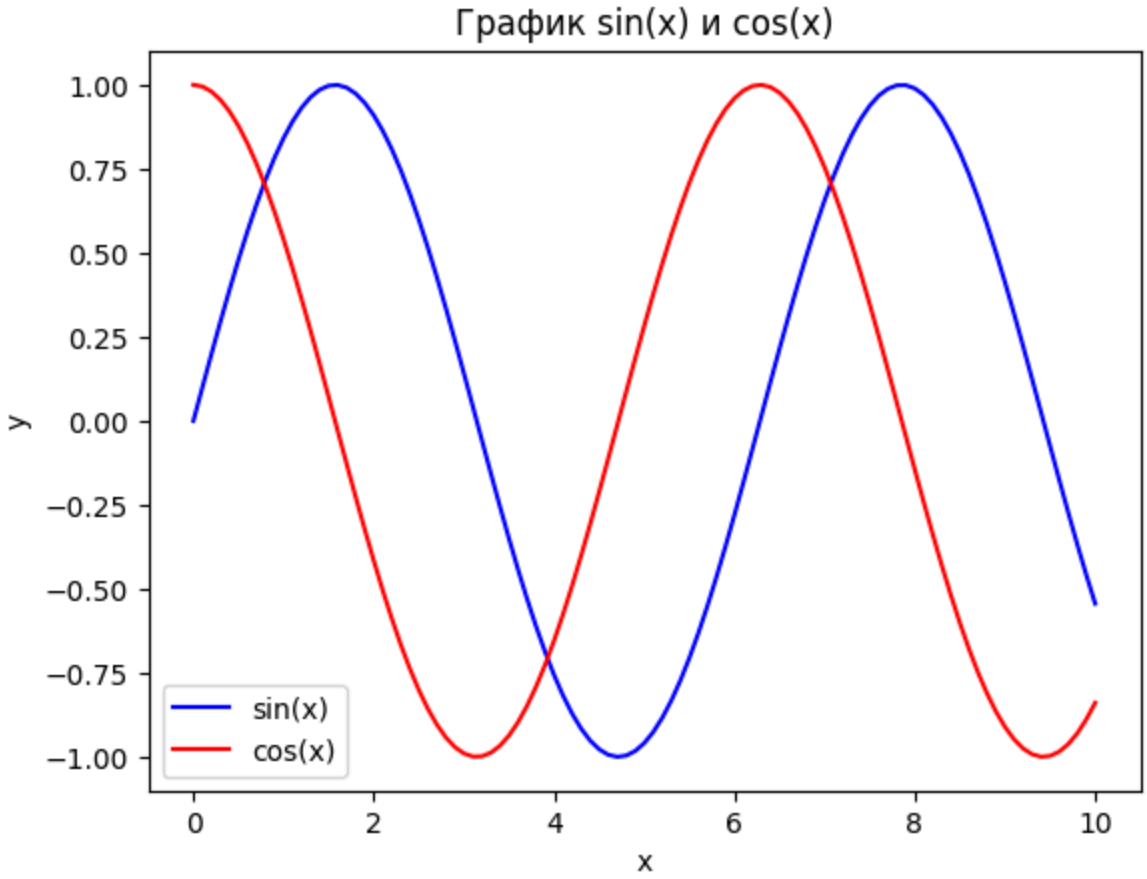
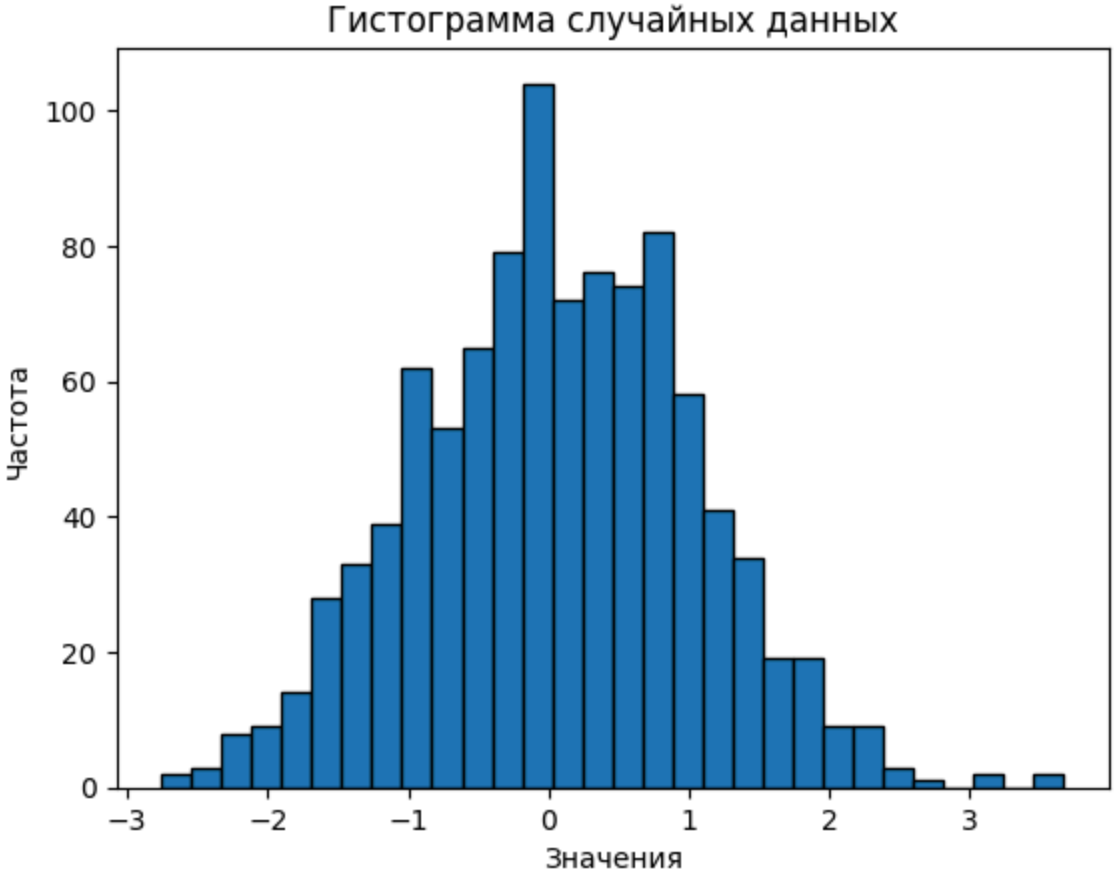


График sin(x) и cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0
2
4
6
8
10
x
sin(x)
cos(x)

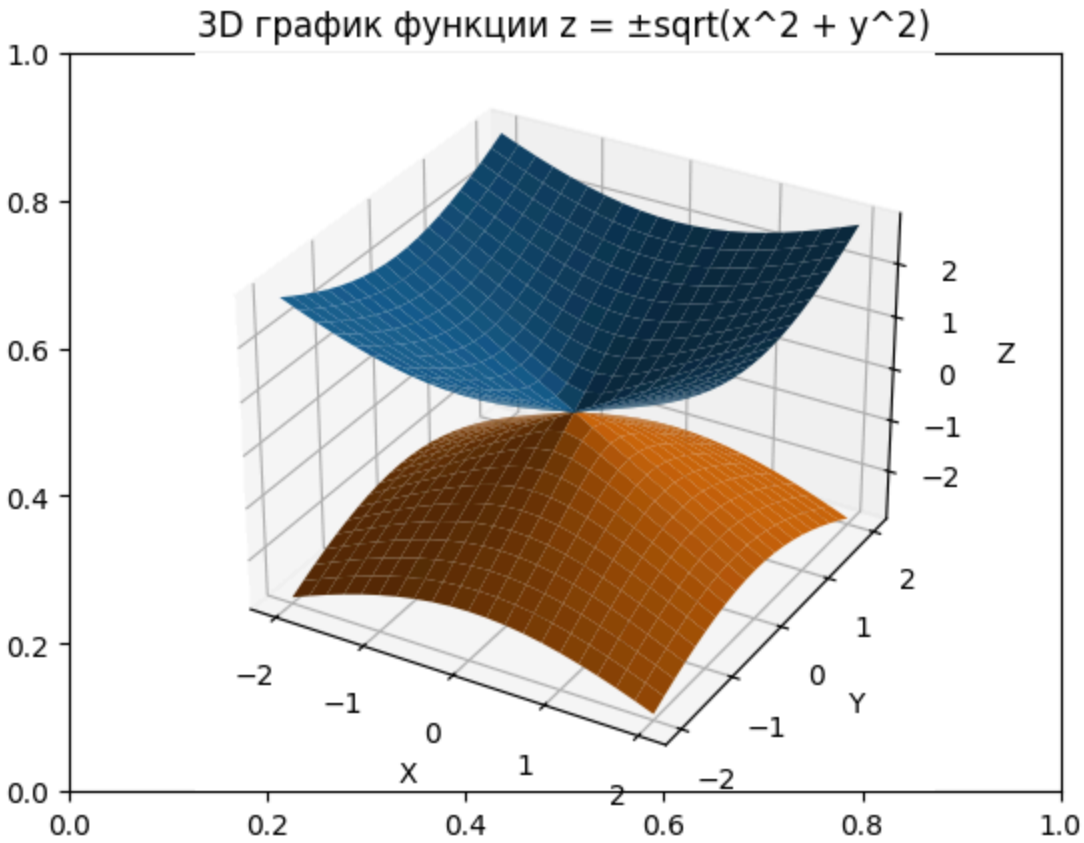

Гистограмма случайных данных
100
80
60
40
20
0
Частота
−3
−2
−1
0
1
2
3
Значения

В Julia для построения 3D-графиков с использованием библиотеки PyPlot используется функция **`surf`**

```
surf(x, y, z)
```

Задача 4. Построить график функции $z(x, y) = \pm\sqrt{x^2 + y^2}$

In [80]:
```
using PyPlot

# Генерация данных для x и y
x = -2:0.2:2
y = -2:0.2:2

# Создаем матрицы X, Y для сетки
X = [xi for xi in x, yi in y]
Y = [yi for xi in x, yi in y]

# Вычисляем значения Z и Z1 для функции z(x, y) = ±(sqrt(x^2 + y^2))
Z = [sqrt(xi^2 + yi^2)  for (xi, yi) in zip(X, Y)]  # Положительная часть
Z1 = [-sqrt(xi^2 + yi^2)  for (xi, yi) in zip(X, Y)]  # Отрицательная часть

# Строим 3D поверхности для положительной и отрицательной части
surf(X, Y, Z)  # Положительная часть
surf(X, Y, Z1)  # Отрицательная часть

# Подписываем оси и добавляем заголовок
xlabel("X")
ylabel("Y")
zlabel("Z")
title("3D график функции z = ±sqrt(x^2 + y^2)")
```

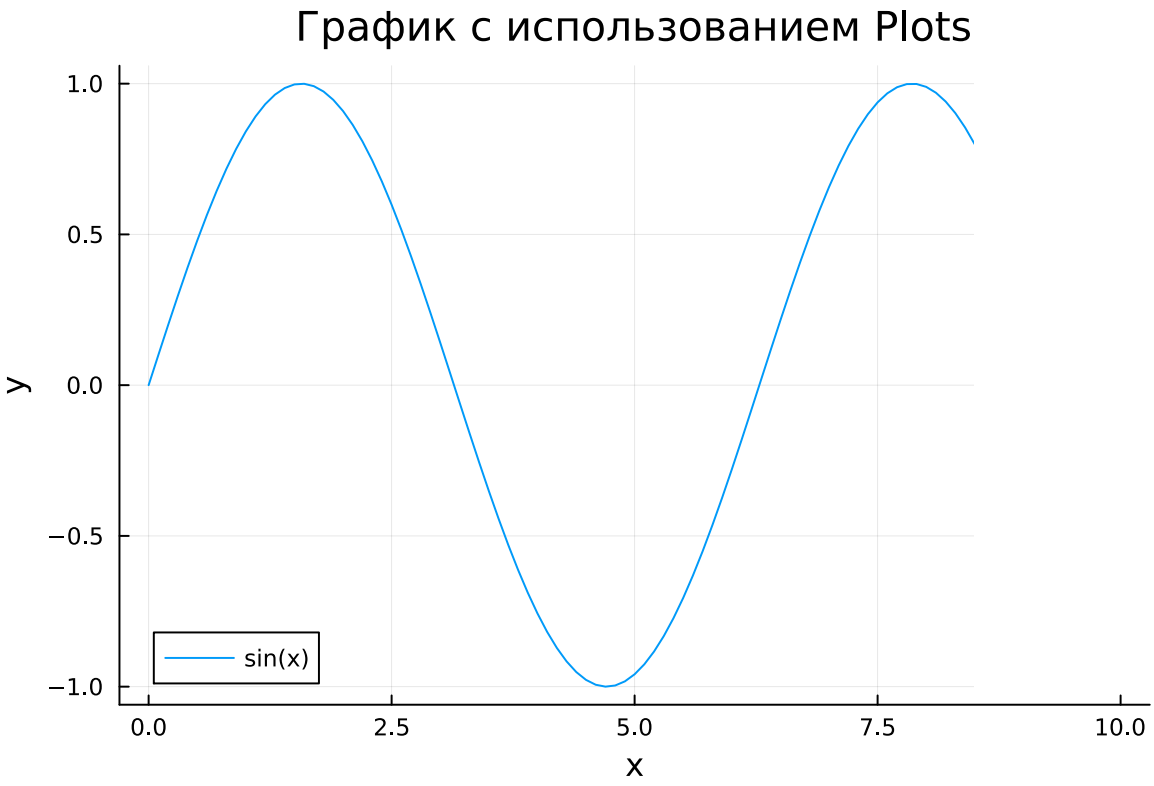


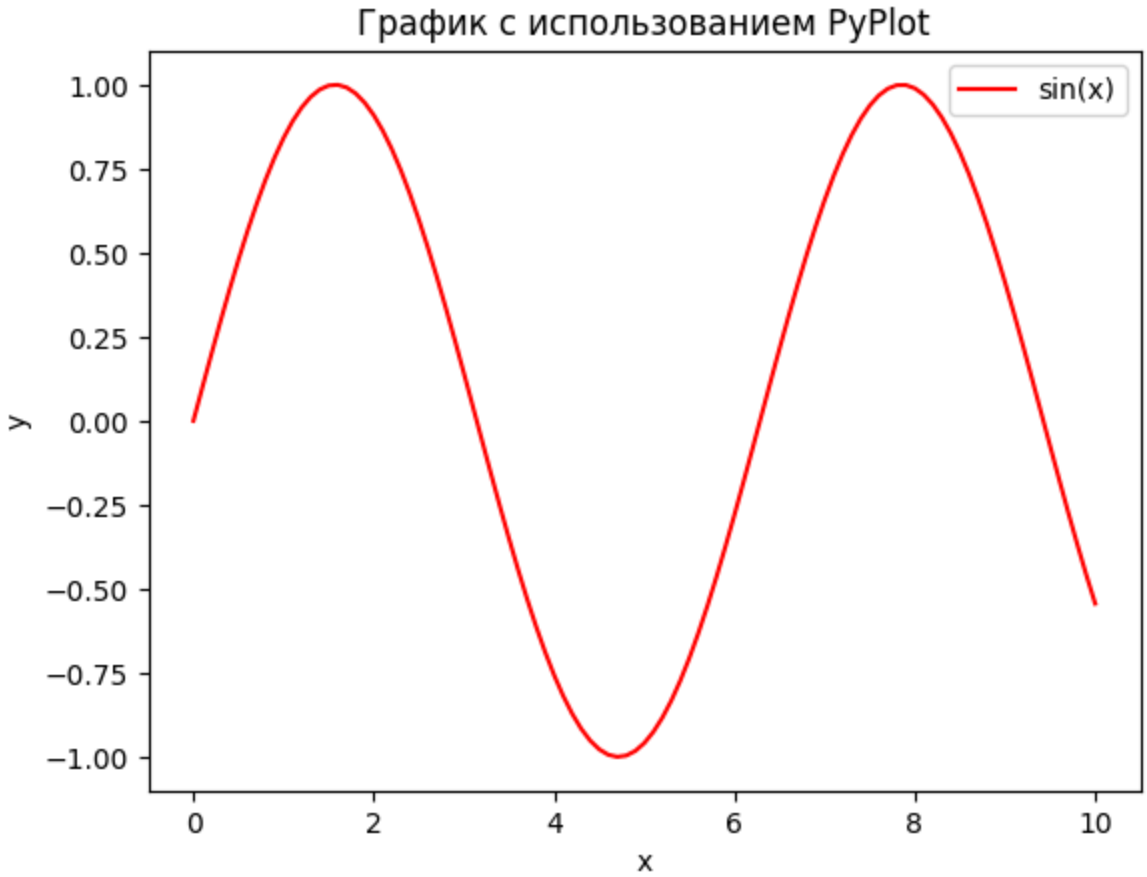
Out[80]: PyObject Text(0.5, 1.0, '3D график функции z = ±sqrt(x^2 + y^2)')

Чтобы использовать **Plots** и **PyPlot** в одном коде, важно помнить, что обе эти библиотеки предоставляют функции с одинаковыми именами. Чтобы избежать конфликта имен, Julia предоставляет возможность явно указывать, из какой библиотеки или модуля должна быть вызвана конкретная функция. Для этого достаточно написать имя библиотеки перед функцией, разделив их точкой. Например, если вы хотите использовать функцию `plot` из библиотеки **Plots**, пишите `Plots.plot()`, а для **PyPlot** — `PyPlot.plot()`.

In [2]:
```julia
using Plots   # Подключаем библиотеку Plots
using PyPlot  # Подключаем библиотеку PyPlot

x = 0:0.1:10
y = sin.(x)

# Строим график с использованием Plots.plot
p1 = Plots.plot(x, y, label="sin(x)", title="График с использованием Plots",

# Строим график с использованием PyPlot.plot
p2 = PyPlot.plot(x, y, label="sin(x)", color="red")  # Явно указываем испол
PyPlot.xlabel("x")
PyPlot.ylabel("y")
PyPlot.title("График с использованием PyPlot")
PyPlot.legend()
```

```
# Отображаем оба графика
display(p1)  # Показываем график, построенный с использованием Plots
PyPlot.show()  # Показываем график, построенный с использованием PyPlot
```





Глава 9. Визуальное программирование

Визуальное программирование — это подход, при котором разработка программ происходит с использованием графических интерфейсов и визуальных элементов. Одним из популярных инструментов для создания таких интерфейсов является **GTK**, который поддерживает различные языки программирования, включая C, Python и Julia. В Julia для работы с GTK используется пакет `Gtk.jl`. Далее мы познакомимся с этим пакетом и его возможностями для создания оконных приложений, добавления кнопок, текстовых полей, панелей и других элементов управления, что позволяет легко создавать интерактивные интерфейсы прямо в Julia.

Виджет — элемент интерфейса для взаимодействия с пользователем (например, кнопка, текстовое поле).

Рендер — процесс преобразования кода в визуальное представление на экране.

9.1. Создание окна (GtkWindow) и кнопки (GtkButton)

Начнём с очень простого примера, в котором создадим пустое окно размером 400x200 пикселей и добавим к нему кнопку. Для этого нам понадобятся несколько ключевых функций из библиотеки **Gtk.jl: GtkWindow, GtkButton, push! и showall**. Давайте подробнее рассмотрим их использование и синтаксис.

1. `GtkWindow` — функция для создания окна с заданными размерами и заголовком.
2. `GtkButton` — это функция для создания кнопки.
3. `push!` — функция для добавления виджетов в окно.
4. `showall` — функция для отображения окна и всех его компонентов.

In [34]:
```
using Gtk

# Создаём окно с заголовком "Моя первая программа на Gtk.jl" и размерами 400
win = GtkWindow("Моя первая программа на Gtk.jl", 400, 200)

# Создаём кнопку с текстом "Нажми на меня"
b = GtkButton("Нажми на меня")

# Добавляем кнопку в окно
push!(win,b)
```

```
# Показываем все элементы окна
showall(win)
```

Out[34]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Моя первая программа на Gt
k.jl", role=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_
NONE, default-width=400, default-height=200, destroy-with-parent=FALSE, hid
e-titlebar-when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GD
K_WINDOW_TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE,
urgency-hint=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, d
eletable=TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to,
has-resize-grip, resize-grip-visible, application, is-active=FALSE, has-top
level-focus=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALS
E, is-maximized=FALSE)
```

Теперь мы расширим пример, чтобы кнопка действительно могла что-то делать.Чтобы сделать кнопку интерактивной (например, вызвать функцию при нажатии), можно добавить обработчик событий с помощью функции **`signal_connect`**. Синтаксис: **`signal_connect(handler, widget, signal_name)`**

Параметры:

1. **`handler`**: Функция (или метод), которая будет вызвана при наступлении события. Эта функция должна принимать как минимум один аргумент — сам виджет, к которому привязан сигнал (например, кнопка).
2. **`widget`**: Виджет (например, кнопка, окно, текстовое поле и т. д.), к которому привязывается обработчик событий.
3. **`signal_name`**: Строка, указывающая на имя события, на которое мы хотим реагировать. Например, "clicked" для кнопки, "changed" для текстового поля, "destroy" для окна.

Напишем программу, которая при нажатии на кнопку будет выводить сообщение в консоль:

In [35]:
```
using Gtk

win = GtkWindow("Пример подключения обработчика событий", 400, 200)

b = GtkButton("Нажми на меня")
push!(win, b)
```

```
# Определяем функцию-обработчик, которая будет вызвана при нажатии на кнопку
function on_button_clicked(b)
    # Эта строка выводит сообщение в консоль, когда кнопка нажата
    println("Кнопка была нажата")
end

# Связываем сигнал "clicked" (событие нажатия на кнопку) с функцией-обработч
# signal_connect устанавливает, что при нажатии на кнопку b будет вызвана фу
signal_connect(on_button_clicked, b, "clicked")

showall(win)
```

Out[35]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример подключения обработ
чика событий", role=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_
WIN_POS_NONE, default-width=400, default-height=200, destroy-with-parent=FA
LSE, hide-titlebar-when-maximized=FALSE, icon, icon-name=NULL, screen, type
-hint=GDK_WINDOW_TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint
=FALSE, urgency-hint=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated
=TRUE, deletable=TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attac
hed-to, has-resize-grip, resize-grip-visible, application, is-active=FALSE,
has-toplevel-focus=FALSE, startup-id, mnemonics-visible=FALSE, focus-visibl
e=FALSE, is-maximized=FALSE)
```

Функция **`set_gtk_property!`** в Gtk.jl используется для динамического изменения внешнего вида и поведения различных Gtk-виджетов во время работы приложения. Все виджеты в Gtk имеют набор свойств, которые можно изменять, например, текст, цвет, размер и поведение. Синтаксис функции **`set_gtk_property!`**:

```
set_gtk_property!(widget, property, value)
```

- widget — это Gtk-виджет, для которого вы хотите установить свойство.
- property — это имя свойства, которое вы хотите изменить. Оно передается как символ, например, :label, :text, :visible и т. д.
- value — значение, которое вы хотите установить для этого свойства. Тип значения зависит от типа свойства.

Пример 1. Изменение текста в кнопке.

Каждому виджету можно установить различные свойства. Например, чтобы изменить текст кнопки, используем свойство `:label`:

In [36]:
```
using Gtk


win = GtkWindow("Изменение текста на кнопке", 400, 200)
b = GtkButton("Нажми на меня")

push!(win, b)

function on_button_clicked(b)
    # Изменяем текст на кнопке с помощью set_gtk_property!
    set_gtk_property!(b, :label, "Нажато")
end

signal_connect(on_button_clicked, b, "clicked")


showall(win)
```

Out[36]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Изменение текста на кнопк
e", role=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NON
E, default-width=400, default-height=200, destroy-with-parent=FALSE, hide-t
itlebar-when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_W
INDOW_TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urg
ency-hint=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, dele
table=TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has
-resize-grip, resize-grip-visible, application, is-active=FALSE, has-toplev
el-focus=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, i
s-maximized=FALSE)
```

Пример 2. Изменение видимости виджета.

Некоторые свойства управляют видимостью или состоянием виджетов. Например, чтобы скрыть или показать виджет, можно использовать свойство `:visible`:

In [37]:
```
using Gtk

win = GtkWindow("Пример наглядности", 400, 200)


b = GtkButton("Нажми на меня")
push!(win, b)

function on_button_clicked(b)
    set_gtk_property!(b, :visible, false)  # Скрываем кнопку
```

```
end

signal_connect(on_button_clicked, b, "clicked")

showall(win)
```

Out[37]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример наглядности", role=
NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, defaul
t-width=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-w
hen-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYP
E_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint
=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRU
E, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-g
rip, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=
FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximiz
ed=FALSE)
```

Пример 3. Изменение размера окна.

Размер окна можно установить с помощью свойств `:width_request` и `:height_request`:

In [38]:
```
using Gtk

win = GtkWindow("Пример размера")


b = GtkButton("Нажми на меня")
push!(win, b)

function on_button_clicked(b)
    set_gtk_property!(win, :width_request, 500)  # Устанавливаем ширину окна
    set_gtk_property!(win, :height_request, 500) # Устанавливаем высоту окна
end

signal_connect(on_button_clicked, b, "clicked")


showall(win)
```

Out[38]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример размера", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-m
aximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HIN
T_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

Пример 4. Изменение состояния кнопки (активна/неактивна)

Можно управлять состоянием кнопки, например, сделать её неактивной с помощью свойства `:sensitive`:

In [39]:
```
using Gtk

win = GtkWindow("Неактивная кнопка")


b = GtkButton("Нажми на меня")
push!(win, b)

function on_button_clicked(b)
    set_gtk_property!(b, :sensitive, false)# Делаем кнопку неактивной (невоз
end

signal_connect(on_button_clicked, b, "clicked")


showall(win)
```

```
Out[39]: GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Неактивная кнопка", role=N
ULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default
-width=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when
-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_H
INT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FA
LSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE,
gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-gri
p, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FA
LSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized
=FALSE)
```

9.2. Макеты

Обычно в приложение требуется добавить более одного виджета. Для этого GTK предоставляет несколько виджетов для организации макета. Вместо использования точного позиционирования, виджеты макета в GTK используют подход, при котором виджеты выравниваются в контейнерах, таких как коробки и таблицы.

9.2.1. GtkBox

`GtkBox` — это один из самых простых и популярных виджетов для создания макета в GTK. Он позволяет организовать другие виджеты в порядке, либо по горизонтали, либо по вертикали. GtkBox упрощает расположение элементов в интерфейсе, избавляя от необходимости вручную управлять позиционированием.

Основные параметры и методы:

1. **Ориентация**: Вы можете указать, как именно вы хотите выравнивать виджеты внутри GtkBox — по горизонтали или по вертикали.

- **`GtkBox(:h)`** — горизонтальное расположение.
- **`GtkBox(:v)`** — вертикальное расположение.

2. **Заполнение и отступы**: Вы можете управлять тем, как виджеты растягиваются внутри GtkBox с помощью параметров expand, fill и padding.

- **`expand`** — указывает, будет ли виджет расширяться, чтобы заполнить доступное пространство.
- **`fill`** — указывает, будет ли виджет растягиваться по размеру в том направлении, в котором он выровнен (горизонтально или вертикально).
- **`padding`** — добавляет отступы вокруг виджета.

3. **Добавление виджетов**: Виджеты добавляются в GtkBox в определённом порядке, и порядок их добавления влияет на то, в какой последовательности они будут отображаться (слева направо или сверху вниз).

In [41]:
```
using Gtk

# Создаем окно
win = GtkWindow("GtkBox пример", 400, 200)

# Создаем горизонтальную коробку (выравнивание по горизонтали)
box = GtkBox(:h)  # :h означает горизонтальное выравнивание

# Создаем несколько кнопок
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")

# Добавляем кнопки в коробку
push!(box, button1, button2, button3)

# Добавляем коробку в окно
push!(win, box)

# Показываем окно
showall(win)
```

Out[41]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="GtkBox пример", role=NULL,
resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-widt
h=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-when-ma
ximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT
_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

После того как виджеты были добавлены в GtkBox, мы можем обратиться к каждому из них по индексу и получить их свойства.

In [42]:
```
length(box)# Получаем количество виджетов в GtkBox
```

Out[42]:
```
3
```

In [43]:
```
get_gtk_property(box[1], :label, String)  # Получаем текст первой кнопки
```

Out[43]:
```
"Кнопка 1"
```

In [44]:
```
get_gtk_property(box[2], :label, String)  # Получаем текст второй кнопки
```

Out[44]:
```
"Кнопка 2"
```

Предположим, что вы хотите, чтобы кнопка "Кнопка 3" заполнила доступное пространство, а между кнопками был отступ. Мы можем настроить свойства expand и spacing для этого:

In [45]:
```
using Gtk


win = GtkWindow("GtkBox пример", 400, 200)

# Создаем горизонтальную коробку (выравнивание по горизонтали)
box = GtkBox(:h)  # :h означает горизонтальное выравнивание

# Создаем несколько кнопок
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")

# Добавляем кнопки в коробку
push!(box, button1, button2, button3)

# Добавляем коробку в окно
push!(win, box)

# Делаем кнопку расширяемой
set_gtk_property!(box, :expand, button3, true)

# Добавляем отступы между виджетами в GtkBox
set_gtk_property!(box, :spacing, 10)

# Показываем окно
showall(win)
```

Out[45]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="GtkBox пример", role=NULL,
resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-widt
h=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-when-ma
ximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT
_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

9.2.2. GtkButtonBox

Если вы хотите, чтобы кнопки в GtkBox имели равный размер и правильное выравнивание, лучше использовать **`GtkButtonBox`**. Этот виджет предназначен специально для создания горизонтальных или вертикальных групп кнопок.

In [46]:
```
using Gtk

win = GtkWindow("Пример GtkButtonBox", 400, 200)

# Создаем GtkButtonBox с горизонтальным расположением
hbox = GtkButtonBox(:h)  # :h — это горизонтальное расположение кнопок

# Создаем кнопки
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")

# Добавляем кнопки в GtkButtonBox
push!(win, hbox)
push!(hbox, button1)
push!(hbox, button2)
push!(hbox, button3)

showall(win)
```

Out[46]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkButtonBox", role
=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, defau
lt-width=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-
when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TY
PE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hin
t=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TR
UE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-
grip, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus
=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximi
zed=FALSE)
```

9.2.3. GtkGrid

`GtkGrid` — это макет, который позволяет размещать другие виджеты в виде сетки с строками и столбцами.

В GtkGrid можно задавать как обычное размещение виджетов, так и задавать их размеры и отступы между ними. Этот макет используется для построения интерфейсов с более сложным расположением элементов, чем, например, вертикальные или горизонтальные контейнеры.

Для добавления виджетов в сетку мы используем индексы строк и столбцов. Например:

In [47]:
```
using Gtk

win = GtkWindow("Пример GtkGrid")
# Создаем сетку
g = GtkGrid()

# Создаем несколько кнопок
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")

# Размещение кнопок в сетке
g[1, 1] = button1  # Кнопка 1 в первом столбце и в первой строке
g[2, 1] = button2  # Кнопка 2 во втором столбце и в первой строке
g[1, 2] = button3  # Кнопка 3 в первом столбце и во второй строке


# Отображаем окно с кнопками
```

```
push!(win, g)
showall(win)
```

Out[47]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkGrid", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-m
aximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HIN
T_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

Свойства GtkGrid:

- `row_spacing` — промежуток между строками.
- `column_spacing` — промежуток между столбцами.
- `column_homogeneous` — если установлено в true, все столбцы будут одинаковой ширины.
- `row_homogeneous` — если установлено в true, все строки будут одинаковой высоты.

In [48]:
```
using Gtk

win = GtkWindow("Пример GtkGrid")
# Создаем сетку
g = GtkGrid()

# Создаем несколько кнопок
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")
button4 = GtkButton("Кнопка 4")

# Размещение виджетов в сетке
g[1, 1] = button1  # button1 в ячейке (1, 1)
g[2, 1] = button2  # button2 в ячейке (2, 1)
g[1, 2] = button3  # button3 в ячейке (1, 2)
g[2, 2] = button4  # button4 в ячейке (2, 2)

# Настройка свойств GtkGrid
set_gtk_property!(g, :row_spacing, 10)         # Промежуток между строками
set_gtk_property!(g, :column_spacing, 15)      # Промежуток между столбцами
set_gtk_property!(g, :column_homogeneous, true) # Все столбцы одинаковой шир
```

```
set_gtk_property!(g, :row_homogeneous, true)    # Все строки одинаковой высо

# Добавляем сетку в окно
push!(win, g)


showall(win)
```

Out[48]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkGrid", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-m
aximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HIN
T_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

Можно задать несколько ячеек для одного виджета:

In [49]:
```
using Gtk

win = GtkWindow("Пример GtkGrid")
# Создаем сетку
g = GtkGrid()

# Создаем несколько кнопок
button1 = GtkButton("Кнопка 1")
button2 = GtkButton("Кнопка 2")
button3 = GtkButton("Кнопка 3")

# Размещение кнопок в сетке
g[1:2, 1] = button1  # button1 займет 2 столбца в 1 строке
g[3, 1:3] = button2  # button2 займет 3 строки в 3 столбце
g[1:3, 4] = button3  # button3 займет 3 столбца в 4 строке

# Настроим расстояние между столбцами и строками
set_gtk_property!(g, :column_spacing, 10)
set_gtk_property!(g, :row_spacing, 10)

# Отображаем окно с кнопками
push!(win, g)
showall(win)
```

Out[49]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkGrid", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-m
aximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HIN
T_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

9.3. Текстовые поля

В GTK на языке Julia два наиболее распространённых виджета для работы с текстом — это **`GtkLabel`** и **`GtkEntry`**. Оба виджета используются для работы с текстом, но их функциональность различна.

9.3.1. GtkLabel

`GtkLabel` — Метка (Текст, который не редактируется)

GtkLabel представляет собой виджет для отображения текста, который не может быть отредактирован пользователем. Это полезно, например, для вывода статического текста, инструкций, заголовков и других элементов, которые должны быть видны, но не изменяемы.

In [50]:
```
using Gtk

# Создаём окно
win = GtkWindow("Пример GtkLabel", 400, 200)

# Создаём метку с текстом
label = GtkLabel("Это метка с текстом!")

push!(win, label)
showall(win)
```

Out[50]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkLabel", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-when
-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_H
INT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FA
LSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE,
gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-gri
p, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FA
LSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized
=FALSE)
```

Текст надписи можно изменить с помощью **`GAccessor.text`**.

In [52]:
```
using Gtk


win = GtkWindow("Пример GtkLabel", 400, 200)

label = GtkLabel("Это метка с текстом!")
GAccessor.text(label,"Мой другой текст")

push!(win, label)
showall(win)
```

Out[52]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример GtkLabel", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=400, default-height=200, destroy-with-parent=FALSE, hide-titlebar-when
-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_H
INT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FA
LSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE,
gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-gri
p, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FA
LSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized
=FALSE)
```

Метод **`GAccessor.markup`** используется для того, чтобы задать разметку (markup) для текста в виджете GtkLabel. Это позволяет не просто

отображать текст, но и форматировать его, например, сделать части текста жирными, курсивными, добавить гиперссылки или изменить цвет.

Давайте рассмотрим примеры, где будет использована разметка для стилизации текста.

Пример 1: Изменение размера текста (size)

In [53]:
```
using Gtk

# Создаем окно
win = GtkWindow("Изменение размера шрифта", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span size="small">Маленький размер шрифта</span>\n
<span size="medium">Средний размер шрифта</span>\n
<span size="large">Большой размер шрифта</span>\n
<span size="x-large">Очень большой размер шрифта</span>\n
<span size="xx-large">Очень-очень большой размер шрифта</span>
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[53]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Изменение размера шрифта",
role=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, d
efault-width=400, default-height=300, destroy-with-parent=FALSE, hide-title
bar-when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDO
W_TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency
-hint=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletabl
e=TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-res
ize-grip, resize-grip-visible, application, is-active=FALSE, has-toplevel-f
ocus=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-ma
ximized=FALSE)
```

Пример 2: Установка цвета текста (foreground)

In [54]:
```
using Gtk

# Создаем окно
win = GtkWindow("Цвет текста", 400, 300)
```

```
label = GtkLabel("")
GAccessor.markup(label,"""
<span foreground="red">Красный цвет текста</span>\n
<span foreground="green">Зеленый цвет текста</span>\n
<span foreground="blue">Синий цвет текста</span>\n
<span foreground="purple">Пурпурный цвет текста</span>\n
<span foreground="#FF5733">Пользовательский цвет (#FF5733)</span>
""")

push!(win, label)
showall(win)
```

Out[54]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Цвет текста", role=NULL, r
esizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width
=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar-when-max
imized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_
NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE,
accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravi
ty=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, res
ize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, s
tartup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALS
E)
```

Пример 3: Установка фона текста (background)

In [55]:
```
using Gtk

# Создаем окно
win = GtkWindow("Цвет фона текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span background="yellow" foreground="black">Желтый фон и черный текст </spa
<span background="lightgray" foreground="black">Светло-серый фон и черный те
<span background="cyan" foreground="black">Голубой фон и черный текст </spar
<span background="black" foreground="white">Черный фон и белый текст</span>
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[55]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Цвет фона текста", role=NU
LL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-
width=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar-whe
n-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_
HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=F
ALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE,
gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-gri
p, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FA
LSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized
=FALSE)
```

Пример 4: Использование конкретного шрифта (font)

In [56]:
```
using Gtk

# Создаем окно
win = GtkWindow("Шрифт текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span font="Arial 14">Шрифт Arial, размер 14</span>\n
<span font="Courier New 18">Шрифт Courier New, размер 18</span>\n
<span font="Times New Roman 20">Шрифт Times New Roman, размер 20</span>\n
<span font="Helvetica 16">Шрифт Helvetica, размер 16</span>
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[56]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Шрифт текста", role=NULL,
resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-widt
h=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar-when-ma
ximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT
_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

Пример 5: Изменение жирности шрифта (weight)

In [57]:
```
using Gtk

# Создаем окно
win = GtkWindow("Жирность текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span weight="normal">Обычный шрифт</span>\n
<span weight="bold"> Жирный шрифт</span>\n
<span weight="light">Легкий шрифт</span>\n
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[57]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Жирность текста", role=NUL
L, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-w
idth=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar-when
-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_H
INT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FA
LSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE,
gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-gri
p, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FA
LSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized
=FALSE)
```

Пример 6: Изменение стиля шрифта (style)

In [58]:
```
using Gtk

# Создаем окно
win = GtkWindow("Стиль текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span style="normal">Нормальный стиль</span>\n
<span style="italic">Курсивный стиль</span>\n
<span style="oblique">Наклонный стиль</span>\n
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[58]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Стиль текста", role=NULL,
resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-widt
h=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar-when-ma
ximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT
_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALS
E, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gr
avity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip,
resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALS
E, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=F
ALSE)
```

Пример 7: Подчеркивание текста (underline)

In [59]:
```
using Gtk

# Создаем окно
win = GtkWindow("Подчеркивание текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span underline="none">Без подчеркивания</span>\n
<span underline="single">Одиночное подчеркивание</span>\n
<span underline="double">Двойное подчеркивание</span>\n
<span underline="error">Подчеркивание ошибочного текста</span>
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[59]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Подчеркивание текста", rol
e=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, defa
ult-width=400, default-height=300, destroy-with-parent=FALSE, hide-titlebar
-when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_T
YPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hi
nt=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=T
RUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize
-grip, resize-grip-visible, application, is-active=FALSE, has-toplevel-focu
s=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maxim
ized=FALSE)
```

Пример 8: Перечеркивание текста (strikethrough)

In [60]:
```
using Gtk

# Создаем окно
win = GtkWindow("Перечеркивание текста", 400, 300)

label = GtkLabel("")
GAccessor.markup(label,"""
<span strikethrough="false">Без перечеркивания</span>\n
<span strikethrough="true">Перечеркнутый текст</span>
""")

# Добавляем метку в окно
push!(win, label)

# Показываем окно
showall(win)
```

Out[60]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Перечеркивание текста", ro
le=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, def
ault-width=400, default-height=300, destroy-with-parent=FALSE, hide-titleba
r-when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_
TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-h
int=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=
TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resiz
e-grip, resize-grip-visible, application, is-active=FALSE, has-toplevel-foc
us=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maxi
mized=FALSE)
```

9.3.2. GtkEntry

`GtkEntry` — это виджет для ввода текста, который представляет собой однострочное текстовое поле.

Основные свойства GtkEntry

- **`:text`** — текущее содержимое текстового поля (строка).
- **`:visibility`** — отображение текста (используется, например, для скрытия текста в поле ввода пароля).
- **`:editable`** — разрешает или запрещает редактирование текста.
- **`:placeholder_text`** — текст-подсказка, который отображается в поле, если оно пустое (например, для указания пользователю, что нужно ввести).

Пример 1. Создание простого поля для ввода текста.

In [61]:
```
using Gtk

# Создаем окно
win = GtkWindow("GtkEntry")

# Создаем поле для ввода текста
entry = GtkEntry()

# Устанавливаем начальный текст
set_gtk_property!(entry, :text, "Введите текст сюда")

# Добавляем поле в окно
push!(win, entry)
showall(win)
```

Out[61]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="GtkEntry", role=NULL, resi
zable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=-
1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-maximiz
ed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NORM
AL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, acc
ept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity=
GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resize
-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, star
tup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

Пример 2. Получение текста из поля ввода.

In [62]:
```
using Gtk

# Создаем окно
win = GtkWindow("GtkEntry",400,300)

# Создаем поле для ввода текста
entry = GtkEntry()

# Устанавливаем текст по умолчанию
set_gtk_property!(entry, :placeholder_text, "Введите ваш email")

# Функция для получения введенного текста
function get_input_text(widget)
    str = get_gtk_property(entry,:text,String)
    println("Введенный текст: ", str)
end

# Добавляем обработчик события для нажатия клавиши Enter
signal_connect(get_input_text, entry, "activate")

# Добавляем поле в окно
push!(win, entry)
showall(win)
```

Out[62]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="GtkEntry", role=NULL, resi
zable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=40
0, default-height=300, destroy-with-parent=FALSE, hide-titlebar-when-maximi
zed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NOR
MAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, ac
cept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity
=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resiz
e-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, sta
rtup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

Пример 3. Работа с паролем (скрытие текста).

Для создания поля для ввода пароля, где введенный текст скрывается, можно использовать свойство `:visibility`.

In [63]:
```
using Gtk

# Создаем окно
win = GtkWindow("Password", 500, 400)

# Создаем поле для ввода пароля
entry = GtkEntry()

# Устанавливаем текст-подсказку
set_gtk_property!(entry, :placeholder_text, "Введите ваш пароль")

# Скрываем введенный текст (для поля пароля)
set_gtk_property!(entry, :visibility, false)

# Добавляем поле в окно
push!(win, entry)
showall(win)
```

Out[63]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Password", role=NULL, resi
zable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=50
0, default-height=400, destroy-with-parent=FALSE, hide-titlebar-when-maximi
zed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NOR
MAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, ac
cept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity
=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resiz
e-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, sta
rtup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

Пример 4. Отключение редактирования.

Чтобы запретить пользователю редактировать текст в поле ввода, можно использовать свойство `:editable`:

In [64]:
```
using Gtk

# Создаем окно
win = GtkWindow("Пример записи, не подлежащей редактированию", 500, 400)

# Создаем поле для ввода текста
entry = GtkEntry()

# Устанавливаем начальный текст
set_gtk_property!(entry, :text, "Этот текст нельзя редактировать")

# Отключаем редактирование
set_gtk_property!(entry, :editable, false)

# Добавляем поле в окно
push!(win, entry)
showall(win)
```

Out[64]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример записи, не подлежащ
ей редактированию", role=NULL, resizable=TRUE, modal=FALSE, window-position
=GTK_WIN_POS_NONE, default-width=500, default-height=400, destroy-with-pare
nt=FALSE, hide-titlebar-when-maximized=FALSE, icon, icon-name=NULL, screen,
type-hint=GDK_WINDOW_TYPE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-
hint=FALSE, urgency-hint=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decor
ated=TRUE, deletable=TRUE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, a
ttached-to, has-resize-grip, resize-grip-visible, application, is-active=FA
LSE, has-toplevel-focus=FALSE, startup-id, mnemonics-visible=FALSE, focus-v
isible=FALSE, is-maximized=FALSE)
```

9.4. Виджеты списка и дерева

`GtkTreeView` - это очень мощный виджет для отображения табличных или иерархических данных. Несмотря на название, GtkTreeView используется не только для отображения деревьев, но и для работы с обычными списками. Главной особенностью этих виджетов является то, что GtkTreeView не хранит данные напрямую. Вместо этого данные хранятся в контейнерах, таких как **`GtkListStore`** для списков и **`GtkTreeStore`** для деревьев.

Списки (или таблицы) представляют собой упорядоченные коллекции данных, где каждый элемент можно воспринимать как строку. Например, можно создать таблицу, в которой будут храниться данные о людях: имя, возраст и пол. Для таких данных удобно использовать GtkListStore, который представляет собой таблицу, где каждая строка может содержать несколько значений разных типов. Если же необходимо отобразить данные в виде дерева, где элементы могут иметь дочерние элементы, то лучше использовать GtkTreeStore. Дерево позволяет организовать элементы в иерархическую структуру, где один элемент может быть родителем других, создавая, например, структуру каталогов в файловой системе. В обоих случаях для отображения данных используется виджет GtkTreeView, который привязывается к контейнеру данных и отображает их на экране. Отличие между списком и деревом заключается в том, что в списке все элементы независимы друг от друга, а в дереве каждый элемент может иметь потомков, создавая иерархию.

9.4.1. GtkListStore

1. **Создание GtkListStore**:

```
store = GtkListStore(T1, T2, ..., Tn)
```

`T1, T2, ..., Tn` — типы данных для каждого столбца. Например, String, Int, Bool, и т.д.

2. **Добавление строки**:

```
push!(store, (value1, value2, ..., value_n))
```

- `store` — объект GtkListStore.
- `value1, value2, ..., value_n` — значения для добавления в строку.

3. **Вставка данных**:

```
insert!(store, row_index, (value1, value2, ..., value_n))
```

- `store` — это объект GtkListStore.
- `row_index` — индекс строки, в которую нужно вставить данные.
- `(value1, value2, ..., value_n)` — кортеж значений, которые будут вставлены в соответствующие столбцы.

4. **Получение данных с использованием индексации**:

```
ls[row_index, column_index]
```

- Возвращает строку по индексу.

```
ls[row_index, column_index] = value
```

- Устанавливает значение в строку и столбец по индексу.

5. **Получение количества строк**:

```
length(store)
```

- Возвращает количество строк в GtkListStore.

6. **GtkTreeView(GtkTreeModel(store))**:

```
GtkTreeView(GtkTreeModel(store))
```

GtkTreeView — это виджет, который отображает данные из модели в виде таблицы или дерева. Важный момент: для отображения данных в GtkTreeView нам нужно передать модель данных через интерфейс GtkTreeModel. GtkTreeView получает данные из модели и использует их для отображения на экране.

7. **Рендереры (Renderers)**:

Рендереры — это компоненты, которые отвечают за визуализацию данных, предоставленных моделью, в виде видимых элементов на экране.

Они могут отображать текст, изображения, прогресс-бары, чекбоксы и другие элементы интерфейса.

- `GtkCellRendererText()`

GtkCellRendererText — это рендерер, который используется для отображения текста в ячейках таблицы или дерева. Когда вы хотите отображать строковые данные (например, имя или возраст), вам нужно использовать этот рендерер.

- `GtkCellRendererToggle()`

GtkCellRendererToggle — это рендерер, который используется для отображения булевых значений в виде чекбоксов. Когда значение в модели данных представляет собой true или false, этот рендерер позволяет пользователю взаимодействовать с этим значением через чекбокс.

- `GtkCellRendererProgress()`

GtkCellRendererProgress — это рендерер, который используется для отображения прогресса выполнения в ячейках таблицы или дерева. Он принимает числовое значение и отображает его как прогресс (например, от 0% до 100%).

8. **GtkTreeViewColumn**:

`GtkTreeViewColumn` — это объект, который определяет, как отображать данные в каждой колонке, ассоциируя их с рендерерами, которые отвечают за визуальное представление данных (например, текст, чекбоксы, прогресс-бары и т.д.).

Пример: Список людей с именем, возрастом и полом.

In [65]:
```
using Gtk

# Создаем контейнер для данных: 3 колонки (имя, возраст, пол)
ls = GtkListStore(String, Int, Bool)

# Заполняем список данными
push!(ls, ("Саша", 20, false))  # Мужчина
push!(ls, ("Рома", 30, false))   # Мужчина
push!(ls, ("Маша", 25, true))    # Женщина

# Создаем виджет TreeView для отображения данных из контейнера
tv = GtkTreeView(GtkTreeModel(ls))

# Создаем рендереры для отображения данных в колонках. Рендереры — это компо
rTxt = GtkCellRendererText()  # Рендерер для текста
rTog = GtkCellRendererToggle()  # Рендерер для чекбокса
```

```
# Создаем колонки для отображения данных
c1 = GtkTreeViewColumn("Имя", rTxt, Dict([("text", 0)]))  # Колонка для имен
c2 = GtkTreeViewColumn("Возраст", rTxt, Dict([("text", 1)]))   # Колонка для
c3 = GtkTreeViewColumn("Пол", rTog, Dict([("active", 2)]))  # Колонка для по
# Добавляем колонки в TreeView

push!(tv, c1, c2, c3)


# Создаем окно для отображения данных
win = GtkWindow(tv, "List View")

# Показываем окно
showall(win)
```

Out[65]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="List View", role=NULL, res
izable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=-
1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-maximiz
ed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NORM
AL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, acc
ept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity=
GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resize
-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, star
tup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

```
for c in [c1, c2, c3]
    GAccessor.resizable(c, true)
end
```

Разбор кода:

```
for c in [c1, c2, c3]:
```

Это цикл, который проходит по списку объектов колонок c1, c2, и c3.

```
GAccessor.resizable(c, true)
```

Внутри цикла вызывается функция GAccessor.resizable. Эта функция задает, может ли колонка быть изменена по ширине пользователем. Параметр true говорит о том, что колонка будет изменяемой по ширине. Если бы был передан параметр false, колонка не могла бы изменять свою ширину.

Мы хотим, чтобы все три колонки c1, c2 и c3 в GtkTreeView могли быть изменены по ширине пользователем. Этот код позволяет задать такую возможность для каждой из них.

Пример: Список людей с именем, возрастом и полом, с возможностью изменять ширину колонок.

In [66]:
```
using Gtk

# Создаем контейнер для данных: 3 колонки (имя, возраст, пол)
ls = GtkListStore(String, Int, Bool)

# Заполняем список данными
push!(ls, ("Саша", 20, false))  # Мужчина
push!(ls, ("Рома", 30, false))   # Мужчина
push!(ls, ("Маша", 25, true))    # Женщина

# Создаем виджет TreeView для отображения данных из контейнера
tv = GtkTreeView(GtkTreeModel(ls))

# Создаем рендереры для отображения данных в колонках. Рендереры — это компо
rTxt = GtkCellRendererText()  # Рендерер для текста
rTog = GtkCellRendererToggle()  # Рендерер для чекбокса

# Создаем колонки для отображения данных
c1 = GtkTreeViewColumn("Имя", rTxt, Dict([("text", 0)]))  # Колонка для имен
c2 = GtkTreeViewColumn("Возраст", rTxt, Dict([("text", 1)]))   # Колонка для
c3 = GtkTreeViewColumn("Пол", rTog, Dict([("active", 2)]))  # Колонка для по
# Добавляем колонки в TreeView

push!(tv, c1, c2, c3)

for c in [c1, c2, c3]
    GAccessor.resizable(c, true)
end

# Создаем окно для отображения данных
win = GtkWindow(tv, "List View")

# Показываем окно
showall(win)
```

Out[66]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="List View", role=NULL, res
izable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=-
1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-maximiz
ed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NORM
AL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, acc
ept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity=
GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resize
-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, star
tup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

Пример. Отображение прогресса задач.

In [67]:
```
using Gtk

# Создаём модель данных для имени и прогресса
ls = GtkListStore(String, Int)

# Добавляем данные в модель (имя и прогресс)
push!(ls, ("Задача 1", 50))  # Прогресс 50%
push!(ls, ("Задача 2", 80))  # Прогресс 80%
push!(ls, ("Задача 3", 30))  # Прогресс 30%

# Создаём виджет GtkTreeView с моделью данных
tv = GtkTreeView(GtkTreeModel(ls))

# Создаём рендерер для отображения прогресса
rProg = GtkCellRendererProgress()

# Создаём колонку для отображения прогресса
c1 = GtkTreeViewColumn("Задача", GtkCellRendererText(), Dict([("text", 0)]))
c2 = GtkTreeViewColumn("Прогресс", rProg, Dict([("value", 1)]))  # "value" у

# Добавляем колонки в GtkTreeView
push!(tv, c1, c2)

# Создаём окно и отображаем виджет
win = GtkWindow(tv, "Таблица с прогрессом")
showall(win)
```

Out[67]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Таблица с прогрессом", rol
e=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, defa
ult-width=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-w
hen-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYP
E_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint
=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRU
E, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-g
rip, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus=
FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximiz
ed=FALSE)
```

Когда мы работаем с таблицами или списками данных в GTK, часто возникает необходимость взаимодействовать с выбранным элементом. Для этого используется объект **`GtkTreeSelection`**, который позволяет отслеживать и управлять выбором элементов в списке. Чтобы получить этот объект, необходимо использовать функцию:

```
selection = GAccessor.selection(tv)
```

Здесь **`tv`** — это объект **`GtkTreeView`**, в котором отображаются данные. С помощью **`GtkTreeSelection`** мы можем задать режим выбора: один элемент или несколько. В данном примере мы будем использовать выбор только одного элемента (одиночный выбор). Для включения мульти-выбора можно вызвать:

```
selection = GAccessor.mode(selection,
Gtk.GConstants.GtkSelectionMode.MULTIPLE)
```

Для текущего примера мы будем использовать одиночный выбор, и нам нужно получить индекс выбранного элемента. Это можно сделать следующим образом:

```
selected_item = selected(selection)
println("Выбранный элемент: ", ls[selected_item, 1])
```

Здесь **`selected(selection)`** возвращает индекс выбранного элемента, и мы можем получить данные из соответствующей строки с помощью индексации.

В случае, если пользователь выбрал элемент и вы хотите выполнить какое-то действие, например, вывести информацию о выбранной строке, можно использовать сигнал **`"changed"`**, который срабатывает каждый раз при изменении выбора:

```
signal_connect(selection, "changed") do widget
  if hasselection(selection)
    currentIt = selected(selection)
    println("Имя: ", ls[currentIt, 1], " Возраст: ", ls[currentIt,
2])
  end
end
```

Этот код подключает обработчик события, который будет вызываться каждый раз, когда пользователь выбирает новый элемент. В нем проверяется, был ли выбран элемент (с помощью `hasselection(selection)`), и если да — выводится информация о выбранной строке.

do widget— это особый синтаксис языка Julia, который используется для создания анонимных функций. Этот синтаксис позволяет нам не объявлять функцию заранее, а сразу передать код, который будет выполняться, когда событие сработает.

В нашем случае do widget — это обработчик события, который будет выполнен, когда сигнал "changed" сработает.

In [68]:
```
using Gtk

# Создаем контейнер для данных: 3 колонки (имя, возраст, пол)
ls = GtkListStore(String, Int, Bool)

# Заполняем список данными
push!(ls, ("Саша", 20, false))  # Мужчина
push!(ls, ("Рома", 30, false))   # Мужчина
push!(ls, ("Маша", 25, true))    # Женщина

# Создаем виджет TreeView для отображения данных из контейнера
tv = GtkTreeView(GtkTreeModel(ls))

# Создаем рендереры для отображения данных в колонках. Рендереры – это компо
rTxt = GtkCellRendererText()  # Рендерер для текста
rTog = GtkCellRendererToggle()  # Рендерер для чекбокса

# Создаем колонки для отображения данных
c1 = GtkTreeViewColumn("Имя", rTxt, Dict([("text", 0)]))  # Колонка для имен
c2 = GtkTreeViewColumn("Возраст", rTxt, Dict([("text", 1)]))   # Колонка для
c3 = GtkTreeViewColumn("Пол", rTog, Dict([("active", 2)]))  # Колонка для по
# Добавляем колонки в TreeView

push!(tv, c1, c2, c3)

# Создаем окно для отображения данных
win = GtkWindow(tv, "List View")


selection = GAccessor.selection(tv)
signal_connect(selection, "changed") do widget
```

```
  if hasselection(selection)
    currentIt = selected(selection)
    println("Выбранный элемент: ", ls[currentIt, 1], " Возраст: ", ls[currer
  end
end

# Показываем окно
showall(win)
```

Out[68]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="List View", role=NULL, res
izable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width=-
1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-when-maximiz
ed=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_NORM
AL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE, acc
ept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravity=
GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, resize
-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, star
tup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALSE)
```

9.4.2. GtkTreeStore

Пример: Создание дерева с использованием GtkTreeStore

In [69]:
```
using Gtk

# Создание модели дерева, в которой будут храниться строки
ts = GtkTreeStore(String)

# Добавляем элементы в дерево
iter1 = push!(ts, ("1",))
iter2 = push!(ts, ("2",), iter1)
iter3 = push!(ts, ("3",), iter2)

# Создаем виджет для отображения дерева
tv = GtkTreeView(GtkTreeModel(ts))

# Создаем рендерер текста для отображения текста в столбце
r1 = GtkCellRendererText()

# Создаем колонку для отображения данных
c1 = GtkTreeViewColumn("A", r1, Dict([("text", 0)]))

# Добавляем колонку в TreeView
push!(tv, c1)

# Создаем окно с TreeView
```

```
win = GtkWindow(tv, "Tree View")

# Показываем окно
showall(win)

# Изменение текста в первом элементе дерева
iter = Gtk.iter_from_index(ts, [1])
ts[iter, 1] = "один"
```

Out[69]: "один"

9.5. События клавиш

Чтобы обрабатывать события нажатия клавиш, необходимо использовать событие **`key-press-event`** для активного окна. Это событие срабатывает каждый раз, когда пользователь нажимает клавишу на клавиатуре в пределах этого окна. Ниже приведен пример кода, который демонстрирует, как это сделать.

In [72]:
```
using Gtk
win = GtkWindow("Пример нажатия клавиш")

# Подключаем обработчик события нажатия клавиши
signal_connect(win, "key-press-event") do widget, event
  # Извлекаем значение нажатой клавиши
  k = event.keyval

  # Выводим на экран код клавиши и сам символ
  println("Вы нажали клавишу с кодом ", k, ", что соответствует символу '",
end

showall(win)
```

Out[72]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Пример нажатия клавиш", ro
le=NULL, resizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, def
ault-width=-1, default-height=-1, destroy-with-parent=FALSE, hide-titlebar-
when-maximized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TY
PE_HINT_NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hin
t=FALSE, accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TR
UE, gravity=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-
grip, resize-grip-visible, application, is-active=FALSE, has-toplevel-focus
=FALSE, startup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximi
zed=FALSE)
```

```
Вы нажали клавишу с кодом 103, что соответствует символу 'g'.
Вы нажали клавишу с кодом 101, что соответствует символу 'e'.
Вы нажали клавишу с кодом 114, что соответствует символу 'r'.
Вы нажали клавишу с кодом 32, что соответствует символу ' '.
```

9.6. Рисование в GTK

С помощью библиотеки Graphics.jl в сочетании с GTK.jl можно рисовать на **канвасе** (Canvas). Канвас (от англ. canvas) в контексте графических интерфейсов — это специальный элемент (виджет), на котором можно рисовать различные графические объекты, такие как линии, прямоугольники, окружности, текст, изображения и так далее.

1. `@GtkCanvas()` Это макрос, создающий канвас для рисования.

Синтаксис:

```
canvas = @GtkCanvas()
```

2. `@guarded draw()` Этот макрос используется для создания обработчика рисования, который будет вызываться каждый раз, когда нужно перерисовать канвас (например, при изменении размеров окна или обновлении содержимого).

Синтаксис:

```
@guarded draw(widget) do widget
    # Код рисования
end
```

- `widget` — это объект, на котором происходит рисование (в данном случае канвас).
- Внутри блока `do` пишется код, который будет выполнен для рисования на канвасе.

3. `getgc()` Функция, которая возвращает контекст рисования для канваса. Контекст рисования — это объект, который управляет такими параметрами, как цвет, шрифт и стиль линии.

Синтаксис:

```
ctx = getgc(canvas)
```

- `canvas` — объект канваса, с которого нужно получить контекст.
- `getgc(canvas)` возвращает объект контекста рисования `ctx`, который используется для выполнения операций рисования, таких как рисование линий, фигур и настройка цвета.

4. **`height()` и `width()`** Эти функции возвращают размеры канваса, которые полезны для адаптивного рисования в зависимости от размеров окна.

Синтаксис:

```
h = height(canvas)
w = width(canvas)
```

- `height(canvas)` — возвращает высоту канваса `canvas`.
- `width(canvas)` — возвращает ширину канваса `canvas`.

5. **`rectangle()`** Рисует прямоугольник в контексте рисования. Этот вызов не заполняет прямоугольник, а только рисует его контур.

Синтаксис:

```
rectangle(ctx, x, y, w, h)
```

- `ctx` — контекст рисования.
- `x`, `y` — координаты верхнего левого угла прямоугольника.
- `w`, `h` — ширина и высота прямоугольника.

6. **`set_source_rgb()`** Задает цвет рисования с использованием модели RGB (красный, зеленый, синий).

Синтаксис:

```
set_source_rgb(ctx, r, g, b)
```

- `ctx` — контекст рисования.
- `r`, `g`, `b` — компоненты красного, зеленого и синего цвета в диапазоне от 0 до 1.

7. **`fill()`** Заполняет текущую форму (например, прямоугольник) текущим цветом.

Синтаксис:

```
fill(ctx)
```
Описание:

- `ctx` — контекст рисования, на котором выполняется операция заполнения.
- Эта функция применяет текущий цвет и заполняет все фигуры, нарисованные до этого (например, прямоугольники, круги).

In [73]:
```
using Gtk, Graphics

# Создаем канвас для рисования
```

```
c = @GtkCanvas()

# Создаем окно с канвасом
win = GtkWindow(c, "Canvas")

# Гарантируем, что рисунок будет перерисован
@guarded draw(c) do widget
    # Получаем контекст рисования (gc) для канваса
    ctx = getgc(c)

    # Получаем размеры канваса
    h = height(c)
    w = width(c)

    # Рисуем первый красный прямоугольник (верхнюю половину окна)
    rectangle(ctx, 0, 0, w, h / 2)
    set_source_rgb(ctx, 1, 0, 0)  # Красный цвет
    fill(ctx)  # Заполняем прямоугольник красным цветом

    # Рисуем второй синий прямоугольник (нижнюю четверть окна)
    rectangle(ctx, 0, 3 * h / 4, w, h / 4)
    set_source_rgb(ctx, 0, 0, 1)  # Синий цвет
    fill(ctx)  # Заполняем прямоугольник синим цветом
end

# Отображаем окно с канвасом
show(c)
```

Out[73]:
```
GtkCanvas(name="", parent, width-request=-1, height-request=-1, visible=TRU
E, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALSE, i
s-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALSE, r
eceives-default=FALSE, composite-child=FALSE, style, events=GDK_POINTER_MOT
ION_MASK | GDK_POINTER_MOTION_HINT_MASK | GDK_BUTTON_PRESS_MASK | GDK_BUTTO
N_RELEASE_MASK | GDK_SCROLL_MASK, no-show-all=FALSE, has-tooltip=FALSE, too
ltip-markup=NULL, tooltip-text=NULL, window, opacity=1,000000, double-buffe
red, halign=GTK_ALIGN_FILL, valign=GTK_ALIGN_FILL, margin-left, margin-righ
t, margin-start=0, margin-end=0, margin-top=0, margin-bottom=0, margin=0, h
expand=FALSE, vexpand=FALSE, hexpand-set=FALSE, vexpand-set=FALSE, expand=F
ALSE, scale-factor=1)
```

Рисование с обработкой событий мыши

Теперь давайте добавим возможность рисовать на канвасе с помощью мыши. Каждый раз, когда пользователь нажимает кнопку мыши (например, левую кнопку), на канвасе будет рисоваться зеленый круг в том месте, где произошел клик.

In [74]:
```
using Gtk, Graphics

# Создаем канвас
c = @GtkCanvas()

# Создаем окно с канвасом
win = GtkWindow(c, "Canvas")
```

```
# Подключаем обработчик события нажатия левой кнопки мыши.(button2press – co
c.mouse.button1press = @guarded (widget, event) -> begin
    # Получаем контекст рисования
    ctx = getgc(widget)

    # Устанавливаем зеленый цвет для рисования
    set_source_rgb(ctx, 0, 1, 0)

    # Рисуем круг в точке, где был клик мышью
    arc(ctx, event.x, event.y, 5, 0, 2pi) # arc(ctx, xc, yc, radius, angle1,
    #ctx: Графический контекст полученный с помощью getgc(widget). Это объек
    #xc, yc: Координаты центра дуги или круга.
    #radius: Радиус дуги или круга.
    #angle1, angle2: Углы, определяющие начало и конец дуги.
    #Углы измеряются в радианах, и если angle1 и angle2 охватывают полный кр
    #Если же углы не охватывают весь круг, будет нарисована дуга.
    stroke(ctx)  # Обводим круг

    # Обновляем канвас, чтобы отобразить изменения
    reveal(widget)
end

# Показываем окно
show(c)
```

Out[74]:
```
GtkCanvas(name="", parent, width-request=-1, height-request=-1, visible=TRU
E, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALSE, i
s-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALSE, r
eceives-default=FALSE, composite-child=FALSE, style, events=GDK_POINTER_MOT
ION_MASK | GDK_POINTER_MOTION_HINT_MASK | GDK_BUTTON_PRESS_MASK | GDK_BUTTO
N_RELEASE_MASK | GDK_SCROLL_MASK, no-show-all=FALSE, has-tooltip=FALSE, too
ltip-markup=NULL, tooltip-text=NULL, window, opacity=1,000000, double-buffe
red, halign=GTK_ALIGN_FILL, valign=GTK_ALIGN_FILL, margin-left, margin-righ
t, margin-start=0, margin-end=0, margin-top=0, margin-bottom=0, margin=0, h
expand=FALSE, vexpand=FALSE, hexpand-set=FALSE, vexpand-set=FALSE, expand=F
ALSE, scale-factor=1)
```

Пример с добавлением прямоугольника и линии.

In [75]:
```
using Gtk

# Создаем канвас
c = @GtkCanvas()

# Создаем окно с канвасом
win = GtkWindow(c, "Canvas")

# Подключаем обработчик события нажатия левой кнопки мыши
c.mouse.button1press = @guarded (widget, event) -> begin
    # Получаем контекст рисования
    ctx = getgc(widget)

    # Рисуем круг с радиусом 20
    set_source_rgb(ctx, 0, 1, 0)  # Зеленый цвет
    arc(ctx, event.x, event.y, 20, 0, 2pi)
    stroke(ctx)  # Обводим круг
```

```
    # Рисуем прямоугольник
    set_source_rgb(ctx, 1, 0, 0)  # Красный цвет
    rectangle(ctx, event.x + 40, event.y + 40, 60, 30) # Команда rectangle(c
    #ctx — графический контекст, полученный через getgc(widget). Это объект,
    #x, y — координаты верхнего левого угла прямоугольника. Эти значения опр
    #width, height — размеры прямоугольника: ширина и высота соответственно.
    stroke(ctx)  # Обводим прямоугольник

    # Рисуем линию
    set_source_rgb(ctx, 0, 0, 1)  # Синий цвет
    move_to(ctx, event.x, event.y)# move_to используется для перемещения кур
    line_to(ctx, event.x + 100, event.y + 100) #line_to(ctx, x, y) использу
    #ctx — графический контекст.
    #x, y — координаты конечной точки линии.

    # Обновляем канвас, чтобы отобразить изменения
    reveal(widget)
end

# Показываем окно
show(c)
```

Out[75]:
```
GtkCanvas(name="", parent, width-request=-1, height-request=-1, visible=TRU
E, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALSE, i
s-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALSE, r
eceives-default=FALSE, composite-child=FALSE, style, events=GDK_POINTER_MOT
ION_MASK | GDK_POINTER_MOTION_HINT_MASK | GDK_BUTTON_PRESS_MASK | GDK_BUTTO
N_RELEASE_MASK | GDK_SCROLL_MASK, no-show-all=FALSE, has-tooltip=FALSE, too
ltip-markup=NULL, tooltip-text=NULL, window, opacity=1,000000, double-buffe
red, halign=GTK_ALIGN_FILL, valign=GTK_ALIGN_FILL, margin-left, margin-righ
t, margin-start=0, margin-end=0, margin-top=0, margin-bottom=0, margin=0, h
expand=FALSE, vexpand=FALSE, hexpand-set=FALSE, vexpand-set=FALSE, expand=F
ALSE, scale-factor=1)
```

9.7. Пример программы

In [1]:
```
using Gtk

# Функция для обновления текста в поле ввода
function update_display(entry, text)
    GAccessor.text(entry, text)
end

# Функция для обработки нажатий кнопок
function on_button_click(entry, label)
    str = get_gtk_property(entry,:text,String)
    new_text = str * label
    update_display(entry, new_text)
end

# Функция для вычисления результата
function calculate_result(entry)
    str = get_gtk_property(entry,:text,String)
    result = eval(Meta.parse(str))
```

```
    str = "$result"
    update_display(entry, str)
end

# Функция для очистки поля ввода
function clear_entry(entry)
    update_display(entry, "")
end

# Основная функция для создания окна калькулятора
function create_calculator_window()
    win = GtkWindow("Калькулятор", 150, 150)

    # Создаем поле для ввода текста (где отображаются числа и операторы)
    entry = GtkEntry()
    set_gtk_property!(entry, :editable, false)
    GAccessor.text(entry, "")
    box = GtkBox(:v)
    push!(box, entry)

    # Создаем сетку для размещения кнопок
    grid = GtkGrid()

    # Создаем кнопки и привязываем обработчики
    button7 = GtkButton("7")
    signal_connect(button7, :clicked) do widget
        on_button_click(entry, "7")
    end

    button8 = GtkButton("8")
    signal_connect(button8, :clicked) do widget
        on_button_click(entry, "8")
    end

    button9 = GtkButton("9")
    signal_connect(button9, :clicked) do widget
        on_button_click(entry, "9")
    end

    button_div = GtkButton("/")
    signal_connect(button_div, :clicked) do widget
        on_button_click(entry, "/")
    end

    button4 = GtkButton("4")
    signal_connect(button4, :clicked) do widget
        on_button_click(entry, "4")
    end

    button5 = GtkButton("5")
    signal_connect(button5, :clicked) do widget
        on_button_click(entry, "5")
    end

    button6 = GtkButton("6")
    signal_connect(button6, :clicked) do widget
```

```
    on_button_click(entry, "6")
end

button_mul = GtkButton("*")
signal_connect(button_mul, :clicked) do widget
    on_button_click(entry, "*")
end

button1 = GtkButton("1")
signal_connect(button1, :clicked) do widget
    on_button_click(entry, "1")
end

button2 = GtkButton("2")
signal_connect(button2, :clicked) do widget
    on_button_click(entry, "2")
end

button3 = GtkButton("3")
signal_connect(button3, :clicked) do widget
    on_button_click(entry, "3")
end

button_sub = GtkButton("-")
signal_connect(button_sub, :clicked) do widget
    on_button_click(entry, "-")
end

button0 = GtkButton("0")
signal_connect(button0, :clicked) do widget
    on_button_click(entry, "0")
end

button_dot = GtkButton(".")
signal_connect(button_dot, :clicked) do widget
    on_button_click(entry, ".")
end

button_eq = GtkButton("=")
signal_connect(button_eq, :clicked) do widget
    calculate_result(entry)
end

button_add = GtkButton("+")
signal_connect(button_add, :clicked) do widget
    on_button_click(entry, "+")
end

# Создание кнопки для очистки поля ввода
button_clear = GtkButton("C")
signal_connect(button_clear, :clicked) do widget
    clear_entry(entry)
end

# Размещение кнопок в сетке
grid[1, 1] = button7
```

```
    grid[2, 1] = button8
    grid[3, 1] = button9
    grid[4, 1] = button_div

    grid[1, 2] = button4
    grid[2, 2] = button5
    grid[3, 2] = button6
    grid[4, 2] = button_mul

    grid[1, 3] = button1
    grid[2, 3] = button2
    grid[3, 3] = button3
    grid[4, 3] = button_sub

    grid[1, 4] = button0
    grid[2, 4] = button_dot
    grid[3, 4] = button_eq
    grid[4, 4] = button_add

    # Размещение кнопки очистки
    grid[1:4, 5] = button_clear


    # Автоматическое подстраивание размеров ячеек
    set_gtk_property!(grid, :column_homogeneous, true)  # Автоматическое под
    set_gtk_property!(grid, :row_homogeneous, true)     # Автоматическое под

    # Добавление кнопок в окно
    push!(box, grid)
    set_gtk_property!(win, :child, box)

    # Показать окно
    showall(win)
end

# Запуск калькулятора
create_calculator_window()
```

```
Gtk-Message: 17:01:36.877: Failed to load module "xapp-gtk3-module"
```

Out[1]:
```
GtkWindowLeaf(name="", parent, width-request=-1, height-request=-1, visible
=TRUE, sensitive=TRUE, app-paintable=FALSE, can-focus=FALSE, has-focus=FALS
E, is-focus=FALSE, focus-on-click=TRUE, can-default=FALSE, has-default=FALS
E, receives-default=FALSE, composite-child=FALSE, style, events=0, no-show-
all=FALSE, has-tooltip=FALSE, tooltip-markup=NULL, tooltip-text=NULL, windo
w, opacity=1,000000, double-buffered, halign=GTK_ALIGN_FILL, valign=GTK_ALI
GN_FILL, margin-left, margin-right, margin-start=0, margin-end=0, margin-to
p=0, margin-bottom=0, margin=0, hexpand=FALSE, vexpand=FALSE, hexpand-set=F
ALSE, vexpand-set=FALSE, expand=FALSE, scale-factor=1, border-width=0, resi
ze-mode, child, type=GTK_WINDOW_TOPLEVEL, title="Калькулятор", role=NULL, r
esizable=TRUE, modal=FALSE, window-position=GTK_WIN_POS_NONE, default-width
=150, default-height=150, destroy-with-parent=FALSE, hide-titlebar-when-max
imized=FALSE, icon, icon-name=NULL, screen, type-hint=GDK_WINDOW_TYPE_HINT_
NORMAL, skip-taskbar-hint=FALSE, skip-pager-hint=FALSE, urgency-hint=FALSE,
accept-focus=TRUE, focus-on-map=TRUE, decorated=TRUE, deletable=TRUE, gravi
ty=GDK_GRAVITY_NORTH_WEST, transient-for, attached-to, has-resize-grip, res
ize-grip-visible, application, is-active=FALSE, has-toplevel-focus=FALSE, s
tartup-id, mnemonics-visible=FALSE, focus-visible=FALSE, is-maximized=FALS
E)
```

Глава 10. Форматированный вывод

Функции **@printf** и **@sprintf** в языке программирования Julia используются аналогично функциям **printf** и **sprintf** из языка Си. Она принимает строку формата, а затем значения, которые нужно вставить в этот формат, и возвращает отформатированную строку.

Вот несколько примеров использования функции **@printf** в Julia:

In [76]:
```
using Printf
x = 3.14159
@printf("%.2f", x)
```

```
3.14
```

In [77]:
```
using Printf
h=-123.678549
k=1234567889
@printf("h=%6.4f\n" ,h)
@printf("k=%15d\n" ,k)
@printf("h=%16.9f\tk=%d\n" , h, k)
```

```
h=-123.6785
k=     1234567889
h=  -123.678549000      k=1234567889
```

In [78]:
```
n = 4
m = 7
A = rand(n, m)

println("Матрица A\n")
for i in 1:n
    for j in 1:m
```

```
        @printf("%6.2f", A[i, j])
    end
    println()
end
```

```
Матрица A

 0.77  0.29  0.05  0.62  0.46  0.70  0.29
 0.86  0.61  0.32  0.16  0.74  0.33  0.26
 0.69  0.82  0.71  0.28  0.34  0.65  0.10
 0.83  0.98  0.86  1.00  0.95  0.70  0.22
```

Глава 11. Матрицы. Решение задач линейной алгебры

Многое, что было рассмотрено ранее для массивов, можно перенести и на обработку матриц. Рассмотрим использование срезов в Julia на примере матриц.

In [79]:
```
n = 4
m = 7
A = rand(n, m)

println("Матрица A\n")
for i in 1:n
    for j in 1:m
        print(A[i, j],"\t")
    end
    println()
end
```

```
Матрица A

0.7073859674769416      0.7711216854719616      0.22868976149558196     0.78
54684782417448  0.3381057840040186      0.31955634437533975     0.0898360801
3734501
0.05643123613867873     0.25849675265551475     0.8909605920691148      0.22
227674601728242 0.1543088443698808      0.49586529601605567     0.6894252428
812635
0.013190935992700048    0.3015576120213943      0.9343886346128013      0.08
142853105761894 0.7549692939570274      0.1363412099801382      0.9614062465
076676
0.20995505658358404     0.4974772279514018      0.4105159504414141      0.03
211045390449996 0.7493336338549643      0.984790933602304       0.4918965426
8737504
```

In [80]:
```
n = 4
m = 7
A = rand(n, m)

println("Матрица A\n")
for i in 1:n
    for j in 1:m
```

```
        @printf("%6.2f", A[i, j])
    end
    println()
end
```

```
Матрица A

 0.38  0.06  0.83  0.84  0.46  0.45  0.48
 0.20  0.55  0.25  0.38  0.72  0.13  0.32
 0.04  0.10  0.72  0.13  0.59  0.09  0.26
 0.99  0.65  0.68  0.13  0.26  0.40  0.32
```

In [81]:
```
b = A[1:3, 3:7]
println(b)
println("Матрица b")
for i in 1:size(b, 1) # Используем size для получения количества строк
    for j in 1:size(b, 2) # Используем size для получения количества столбцо
        @printf("%6.2f", b[i, j])
    end
    println()
end
```

```
[0.8293805482029529 0.844880452883475 0.45848063723328114 0.4515412329982495
5 0.48490297389116377; 0.25107757582558443 0.37803881918509696 0.71912975165
38319 0.13263050359057316 0.32024802395679575; 0.7210912911834485 0.13359053
202443116 0.5912027479814739 0.08751825709861427 0.2601023146629049]
Матрица b
  0.83  0.84  0.46  0.45  0.48
  0.25  0.38  0.72  0.13  0.32
  0.72  0.13  0.59  0.09  0.26
```

In [82]:
```
C = A[1, :]
println(C)
println("Массив C")
for i in 1:size(C, 1)
    @printf("%f ", C[i])
end
```

```
[0.38027129434899065, 0.05788338520995728, 0.8293805482029529, 0.84488045288
3475, 0.45848063723328114, 0.45154123299824955, 0.48490297389116377]
Массив C
0.380271 0.057883 0.829381 0.844880 0.458481 0.451541 0.484903
```

11.1. Решение задач линейной алгебры

Для решения задач линейной алгебры предназначена библиотека LinearAlgebra, среди функций этогй библиотеки есть функции обработки массивов, а также нахождения определителя матрицы, решения системы линейных уравнений, обращения матрицы, нахождение собственных чисел и собственных векторов матрицы др. Некоторые из них представлены в таблице ниже:

Команда	Описание
norm(A)	Функция используется для вычисления нормы матрицы или вектора. Норма вектора задается стандартным способом, а норма матрицы определяется как максимальный сингулярный разрыв матрицы.
det(A)	Функция вычисляет определитель матрицы A.
inv(A)	Функция используется для нахождения обратной матрицы A. Если матрица не обратима, будет выброшено исключение.
eigvals(A)	Функция возвращает собственные значения матрицы A.
eigvecs(A)	Функция возвращает собственные векторы матрицы A.
qr(A)	Функция выполняет QR-разложение матрицы A на ортогональную матрицу Q и верхнюю треугольную матрицу R.
lu(A)	Функция выполняет LU-разложение матрицы A на нижнюю треугольную матрицу L и верхнюю треугольную матрицу U.
svd(A):	Функция выполняет сингулярное разложение матрицы A на матрицы левых и правых сингулярных векторов U и V соответственно, а также диагональную матрицу сингулярных значений Σ.
dot(x, y)	Функция используется для вычисления скалярного произведения двух векторов x и y.
cross(x, y)	Функция используется для вычисления векторного произведения двух трехмерных векторов x и y.
adjoint(A):	Функция возвращает сопряженно-транспонированную матрицу A.
rank(A)	Функция возвращает ранг матрицы A.
trace(A)	Функция возвращает след матрицы A, т.е. сумму элементов главной диагонали.

Задача 1. Написать программу решения системы линейных алгебраический уравнений на языке Julia. Для тестирования программы решения систем линейных алгебраических уравнений будем использовать следующую тестовую систему:

$$A \cdot x = b$$

Матрица $A(N, N)$ формируется следующим образом. Диагональные элеменнты равны $2\dot{N}$, все остальные элементы матрицы равны 1.

Вектор правых частей **b** определяется формулой
$b_i = \frac{n \cdot (n+1)}{2} + i \cdot (2 \cdot (n-1))$, если элементы нумеруются с 1 и
$b_i = \frac{n \cdot (n+1)}{2} + (i+1) \cdot (2 \cdot (n-1))$, если элементы нумеруются с 0.

Матрица A обладает следующими особенностями:

1. Матрица A имеет диагональное преобладание, что позволяет использовать её в качестве тестовой для итерационных методов (Зейделя, Якоби и т. д.).
2. Заранее известно решение системы любой размерности x=(1,2,3,4,5...).

In [84]:
```
using LinearAlgebra

N = parse(Int, readline())
A = ones(N, N)
b = ones(N)

for i in 1:N
    for j in 1:N
        if i == j
            A[i, j] = 2N
        else
            A[i, j] = 1
        end
    end
    b[i] = N  *  (N + 1) / 2 + i  *  (2N - 1)
end

x = A \ b

@printf("Корни уравнения\n")
for i in 1:N
    @printf("%5.2f\t", x[i])
end

r = A  *  x - b

@printf("\nМассив ошибок\n")
for i in 1:N
    @printf("%.15e\t", r[i])
end
```

```
Корни уравнения
 1.00    2.00    3.00    4.00    5.00    6.00    7.00    8.00    9.00   10.0
0       11.00   12.00   13.00
Массив ошибок
0.000000000000000e+00   0.000000000000000e+00   0.000000000000000e+00   -2.8
42170943040401e-14      0.000000000000000e+00   -2.842170943040401e-14  0.00
0000000000000e+00       0.000000000000000e+00   5.684341886080801e-14   5.68
4341886080801e-14       5.684341886080801e-14   5.684341886080801e-14   5.68
4341886080801e-14
```

Задача 2. Написать программу решения системы линейных алгебраический уравнений методом Гаусса на языке Julia. Метод Гаусса оформить в виде функции.

In [85]:

```
# Функция для выполнения метода Гаусса
function gause(A, B, n)
    b = copy(B)
    a = copy(A)

    for k = 1:n
        maxindex = k
        maxval = abs(a[k, k])

        for i = k+1:n
            if abs(a[i, k]) > maxval
                maxval = abs(a[i, k])
                maxindex = i
            end
        end

        for i = k:n
            a[k, i], a[maxindex, i] = a[maxindex, i], a[k, i]
        end
        b[k], b[maxindex] = b[maxindex], b[k]

        for i = k+1:n
            M = a[i, k] / a[k, k]
            for j = k:n
                a[i, j] -= M * a[k, j]
            end
            b[i] -= M * b[k]
        end
    end

        x = zeros(n)
        for i = n:-1:1
            s=0
            for j in i+1:n
            s=s+a[i,j]*x[j];
        end
            x[i] = (b[i] - s) / a[i, i]
        end

    return x
end

println("Введите количество строк (n):")
n = parse(Int, readline())
println("Введите количество столбцов (m):")
m = parse(Int, readline())

println("Введите элементы матрицы A($n x $m):")
A = zeros(Float64, n, m)
for i in 1:n
```

```
    for j in 1:m
   t = parse(Float64, readline())
    A[i,j] = t
end
end

println("Введите элементы массива a($n)")
a = zeros(Float64, n)
for i in 1:n
    k = parse(Float64, readline())
    a[i] = k
end

coef = gause(A, a,n)
```

```
Введите количество строк (n):
Введите количество столбцов (m):
Введите элементы матрицы A(2 x 2):
Введите элементы массива a(2)
```

Out[85]:
```
2-element Vector{Float64}:
  1.875
 -0.25
```

Задача 3. Написать программу найхождения определителя матрицы, вычисления обратной матрицы, нахождения собственные значений и собственные векторов матрицы.

In [86]:
```
using LinearAlgebra
using Printf

N = parse(Int, readline())
A = ones(N, N)

for i in 1:N
    for j in 1:N
        if i == j
            A[i, j] = 2N
        else
            A[i, j] = 1
        end
    end
end

@printf("Исходная матрица A\n");
for i in 1:N
    for j in 1:N
        @printf("%8.2f\t", A[i, j])
    end
    @printf("\n");
end

a1 = inv(A)
@printf("Обратная матрица к матрице A=\n");
for i in 1:N
    for j in 1:N
```

```
        @printf("%8.5f\t", a1[i, j])
    end
    @printf("\n");
end

@printf("Проверка=\n");
a2 = A  *  a1
for i in 1:N
    for j in 1:N
        @printf("%8.5f\t", a2[i, j])
    end
    @printf("\n");
end

dt = det(A)
@printf("Определитель=%8.4f\n", dt)

a3 = eigvals(A)
a4 = eigvecs(A)

@printf("Собственные числа=")
for j in 1:N
    @printf("%8.5f\t", a3[j])
end
@printf("\nСобственные вектора=\n");
for i in 1:N
    for j in 1:N
        @printf("%8.5f\t", a4[i, j])
    end
    @printf("\n");
end
```

Исходная матрица A

```
   20.00           1.00           1.00           1.00           1.00
1.00        1.00           1.00           1.00           1.00
    1.00          20.00           1.00           1.00           1.00
1.00        1.00           1.00           1.00           1.00
    1.00           1.00          20.00           1.00           1.00
1.00        1.00           1.00           1.00           1.00
    1.00           1.00           1.00          20.00           1.00
1.00        1.00           1.00           1.00           1.00
    1.00           1.00           1.00           1.00          20.00
1.00        1.00           1.00           1.00           1.00
    1.00           1.00           1.00           1.00           1.00
20.00       1.00           1.00           1.00           1.00
    1.00           1.00           1.00           1.00           1.00
1.00       20.00           1.00           1.00           1.00
    1.00           1.00           1.00           1.00           1.00
1.00        1.00          20.00           1.00           1.00
    1.00           1.00           1.00           1.00           1.00
1.00        1.00           1.00          20.00           1.00
    1.00           1.00           1.00           1.00           1.00
1.00        1.00           1.00           1.00          20.00
```

Обратная матрица к матрице A=

```
 0.05082        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181         0.05082        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181         0.05082        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181         0.05082        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181         0.05082
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
0.05082 -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181         0.05082        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181         0.05082        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181         0.05082        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181        -0.00181
-0.00181        -0.00181        -0.00181        -0.00181         0.05082
```

Проверка=

```
 1.00000         0.00000        -0.00000        -0.00000        -0.00000
0.00000 -0.00000        -0.00000        -0.00000        -0.00000
-0.00000         1.00000        -0.00000         0.00000        -0.00000
0.00000 -0.00000        -0.00000        -0.00000        -0.00000
-0.00000         0.00000         1.00000         0.00000        -0.00000
0.00000 -0.00000        -0.00000        -0.00000        -0.00000
-0.00000         0.00000         0.00000         1.00000        -0.00000
0.00000  0.00000        -0.00000        -0.00000        -0.00000
-0.00000        -0.00000        -0.00000        -0.00000         1.00000
0.00000 -0.00000        -0.00000        -0.00000        -0.00000
-0.00000        -0.00000        -0.00000        -0.00000        -0.00000
1.00000  0.00000        -0.00000        -0.00000        -0.00000
-0.00000        -0.00000        -0.00000        -0.00000        -0.00000
```

```
0.00000  1.00000        -0.00000        -0.00000        -0.00000
-0.00000         0.00000         0.00000        -0.00000         0.00000
0.00000  0.00000         1.00000         0.00000         0.00000
-0.00000        -0.00000        -0.00000        -0.00000        -0.00000
0.00000 -0.00000         0.00000         1.00000         0.00000
-0.00000        -0.00000        -0.00000        -0.00000        -0.00000
0.00000 -0.00000         0.00000         0.00000         1.00000
Определитель=9357943235591.0039
Собственные числа=19.00000      19.00000        19.00000        19.00000
19.00000        19.00000        19.00000        19.00000        19.00000
29.00000
Собственные вектора=
-0.08571        -0.04801         0.11104         0.17114        -0.28723
-0.12922         0.78413         0.08622         0.35670        -0.31623
-0.57101        -0.04801         0.11104         0.17114        -0.28723
-0.12922        -0.54421         0.08622         0.35670        -0.31623
 0.62372        -0.04801         0.32898        -0.39323        -0.28723
-0.12922        -0.22787        -0.21169         0.22332        -0.31623
 0.27321        -0.04801        -0.24625         0.70732        -0.01881
-0.12922        -0.09981        -0.43094        -0.22224        -0.31623
-0.35628        -0.24869        -0.34164        -0.51353         0.04327
-0.20122         0.13016        -0.45036        -0.26202        -0.31623
-0.07895        -0.04801         0.52490         0.03691        -0.01881
-0.12922         0.02884         0.33887        -0.69417        -0.31623
 0.25266        -0.04801        -0.60962        -0.09563        -0.01881
-0.12922        -0.09231         0.65380        -0.00751        -0.31623
 0.01736        -0.04801         0.19347         0.02399         0.86573
-0.12922        -0.00634         0.02269         0.30439        -0.31623
-0.07501         0.90518        -0.07192        -0.10811         0.00911
0.21281  0.02740        -0.09481        -0.05516        -0.31623
 0.00000        -0.32041         0.00000         0.00000         0.00000
0.89294  0.00000         0.00000         0.00000        -0.31623
```

Глава 12. Файлы в Julia

В Julia можно легко работать с текстовыми файлами, выполняя операции как записи, так и чтения данных. Давайте рассмотрим примеры создания, записи и считывания данных из текстового файла в Julia

Работа с файлом начинается с того, что указывается его имя `fname="test.dat"`. Далее файл нужно открыть `f1=open(fname, mode)`, mode характеризует возможности работы с файлом

mode	Описание
r	Чтение
w	Запись
a	Добавление
r+	Чтение и Запись
w+	Запись и Чтение

mode	Описание
a+	Добавление и Чтение

Можно сразу выполнить команду `f1=open("test.dat","r+")`.

Наиболее простой способ работы с данными из файла с именем fname выглядит так

```
open(fname, mode) do f1
... действия с файлом f1
end
```

В этом случае файл закрывать не нужно, он закрывается автоматически послевыхода из блока.

12.1 Создание и запись данных в файл

Пример текстового файла

In [87]:
```
# Открываем файл для записи (если файл уже существует, он будет перезаписан)
open("file.txt", "w") do file
    write(file, "Привет, мир!\n")
    write(file, "Это текстовый файл, созданный в Julia.")
end
```

Out[87]: 64

In [88]:
```
data = ["Первая строка данных", "Вторая строка данных", "Третья строка данны

open("file.txt", "a") do file  # Открываем файл для добавления данных
    for line in data # Переменная line представляет каждую строку данных из
        write(file, line * "\n")
    end
end
```

Пример двоичного файла

In [89]:
```
data = rand(10)  # генерируем случайные данные
file = open("binary_data.bin", "w")  # открываем файл для записи в двоичном
write(file, data)  # записываем данные в файл
close(file)  # закрываем файл
```

12.2 Чтение данных из файла

Пример текстового файла

Для чтения данных из файла используем функцию eachline, которая читает файл построчно.

In [90]:
```
open("file.txt","r") do file
    for line in eachline(file)
        println(line)
    end
end
```

```
Привет, мир!
Это текстовый файл, созданный в Julia.Первая строка данных
Вторая строка данных
Третья строка данных
```

Пример двоичного файла

In [91]:
```
open("binary_data.bin", "r") do file
    while !eof(file)
        data = read(file, Float64)
        println(data)
    end
end
```

```
0.39020676191782644
0.7085896278547301
0.9121029844444516
0.32804445365394486
0.8610653645025844
0.0020099891151391658
0.6773897812080303
0.6916870536183971
0.3818956079774076
0.02972926828511646
```

Пример. Записать в файл с расширением .bin и считать из него матрицу

$$A_{i,j} = \begin{cases} 2.7 \cdot n, & i = j \\ 1, & i \neq j \end{cases}$$

.

Программа записи в файл

In [92]:
```
print("N = ")
N = parse(Int64, readline())
A = ones(Float64, N, N)
for i in 1:N
    for j in 1:N
        if i == j
            A[i, j] = 2.7 * N
        else
            A[i, j] = 1
        end
    end
end
file = open("matr.bin", "w")
write(file, N)
```

```
write(file, A)
close(file)
```

N =

In [93]:
```
open("matr.bin", "r") do file
    N = read(file, Int)
    while !eof(file)
        for j in 1:N
        for i in 1:N
        data = read(file, Float64)
        print(data," ")
        end
        print("\n")
        end
    end
end
```

```
13.5 1.0 1.0 1.0 1.0
1.0 13.5 1.0 1.0 1.0
1.0 1.0 13.5 1.0 1.0
1.0 1.0 1.0 13.5 1.0
1.0 1.0 1.0 1.0 13.5
```

Глава 13. Решение задач обработки эксперимента

Рассмотрим задачи подбора кривых методом наименьших квадратов (МНК).

Задача 1. В «Основах химии» Д.И. Менделеева приводятся данные о растворимости $NaNO_3$ (P) в зависимости от температуры воды. В 100 частях воды растворяется следующее число условных частей при соответствующих температурах.

T, ^oC	0	4	10	15	21	29	36	51	58
P	66.7	71	76.3	80.6	85.7	92.9	99.4	113.6	125.1

Температура 32°C не входит в наблюдаемые значения. Необходимо определить, какова будет растворимость $NaNO_3$ при соответствующих температурах.

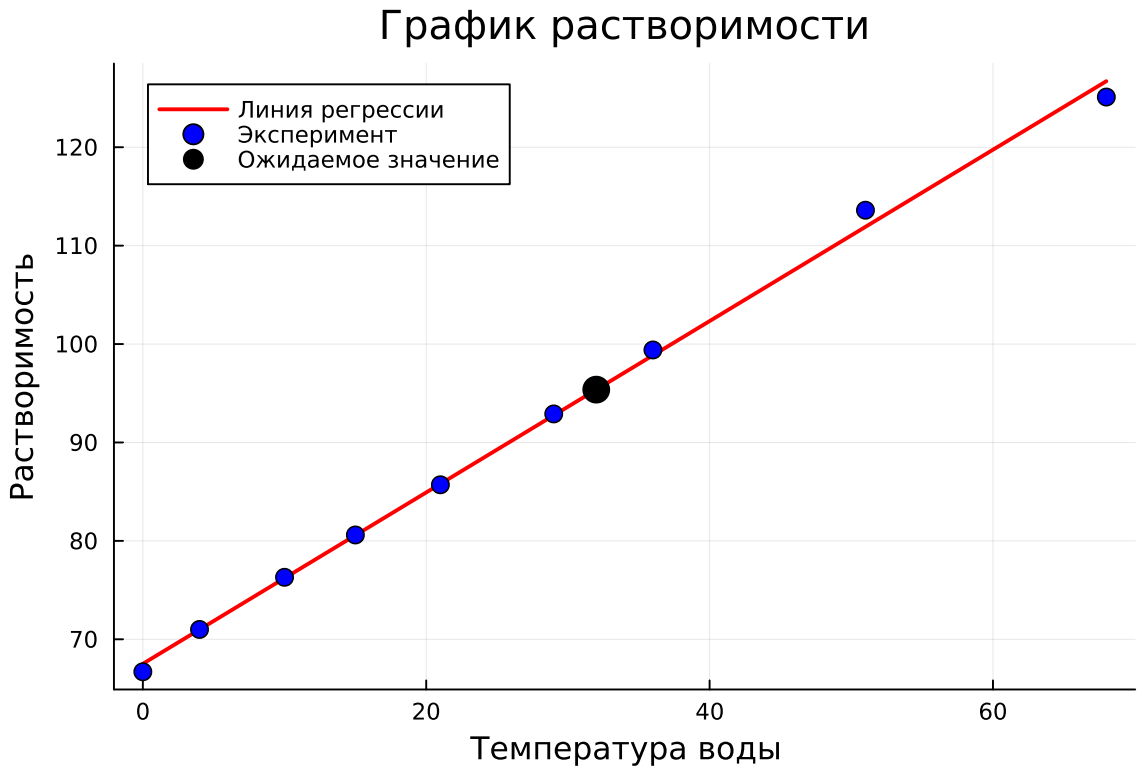
Ниже приведен код программы решения задачи

In [95]:
```
T = [0, 4, 10, 15, 21, 29, 36, 51, 68]
P = [66.7, 71, 76.3, 80.6, 85.7, 92.9, 99.4, 113.6, 125.1]
n = length(T)
Mt = sum(T) / n
Mp = sum(P) / n
b = (n * sum(T .* P) - sum(P) * sum(T)) / (n * sum(T .* T) - sum(T)^2)
```

```
a = Mp - b * Mt
Tl = [T[1], T[end]]
Pl = [a + b * Tl[i] for i in 1:2]
println(Tl)
println(Pl)
T1 = 32
P1 = a + b * T1
println(T1," ", P1)

plot(Tl, Pl, color = :red, linewidth = 2, label = "Линия регрессии")
scatter!(T, P, color = :blue, markersize = 5, label = "Эксперимент")
scatter!([T1], [P1], color = :black, markersize = 8, label = "Ожидаемое знач
title!("График растворимости")
xlabel!("Температура воды")
ylabel!("Растворимость")
```

```
[0, 68]
[67.50779419813904, 126.71134099616856]
32 95.36828680897646
```

Out[95]:



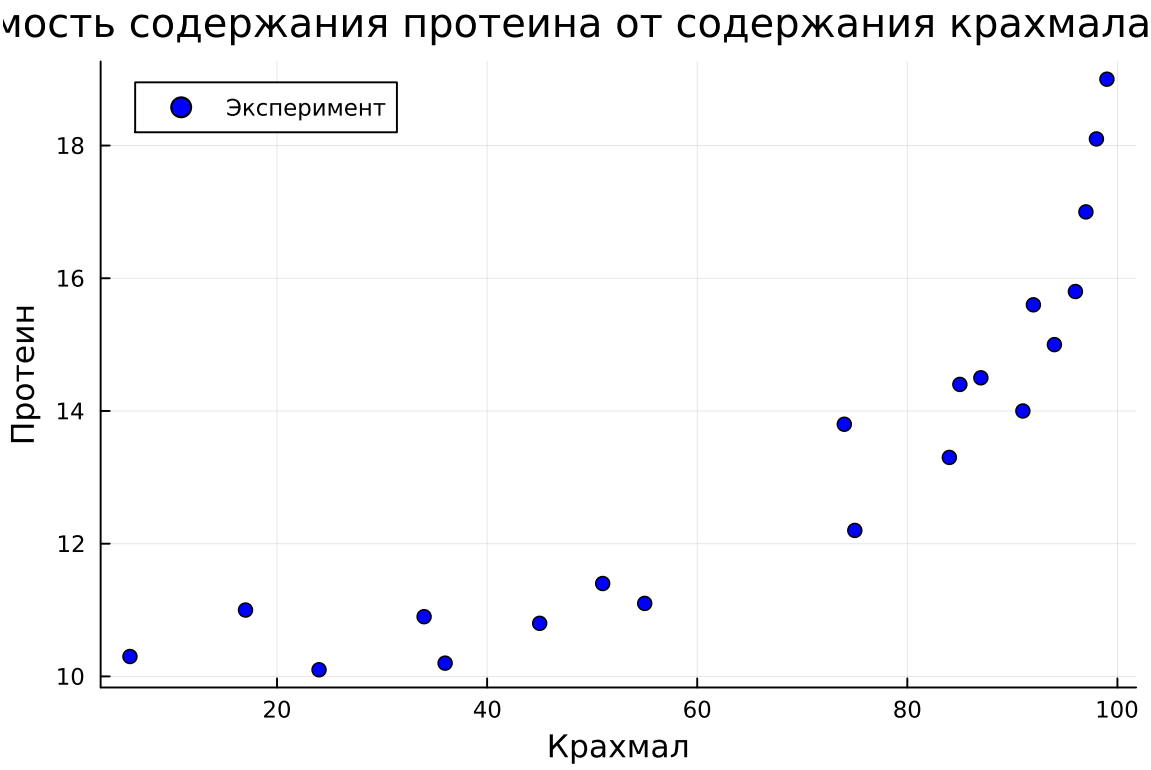
Задача 2. Производится наблюдение над двумя переменными – процентным содержанием протеина (P) и крахмала (K) в зернах пшеницы. Обе переменные характеризуют качество пшеницы, но определение протеина требует сложного химического анализа, а определение крахмала может быть сделано гораздо проще. В таблице приведены результаты 20 наблюдений. Необходимо произвести выравнивание этих наблюдений по квадратичной и по кубической параболам, а затем сравнить результаты полученных вычислений.

P,%	10.3	12.2	14.5	11.1	10.9	18.1	14	10.8	11.4	11	10.2	17	13.8	10.1
K,%	6	75	87	55	34	98	91	45	51	17	36	97	74	24

Напишем небольшой код, который позволяет построить график экспериментальных точек.

In [96]:
```
using Plots

P = [10.3, 12.2, 14.5, 11.1, 10.9, 18.1, 14.0, 10.8, 11.4, 11, 10.2, 17, 13.
K = [6, 75, 87, 55, 34, 98, 91, 45, 51, 17, 36, 97, 74, 24, 85, 96, 92, 94,
scatter(K, P, label="Эксперимент", color = :blue, title = "Зависимость содер
xlabel!("Крахмал")
ylabel!("Протеин")
```

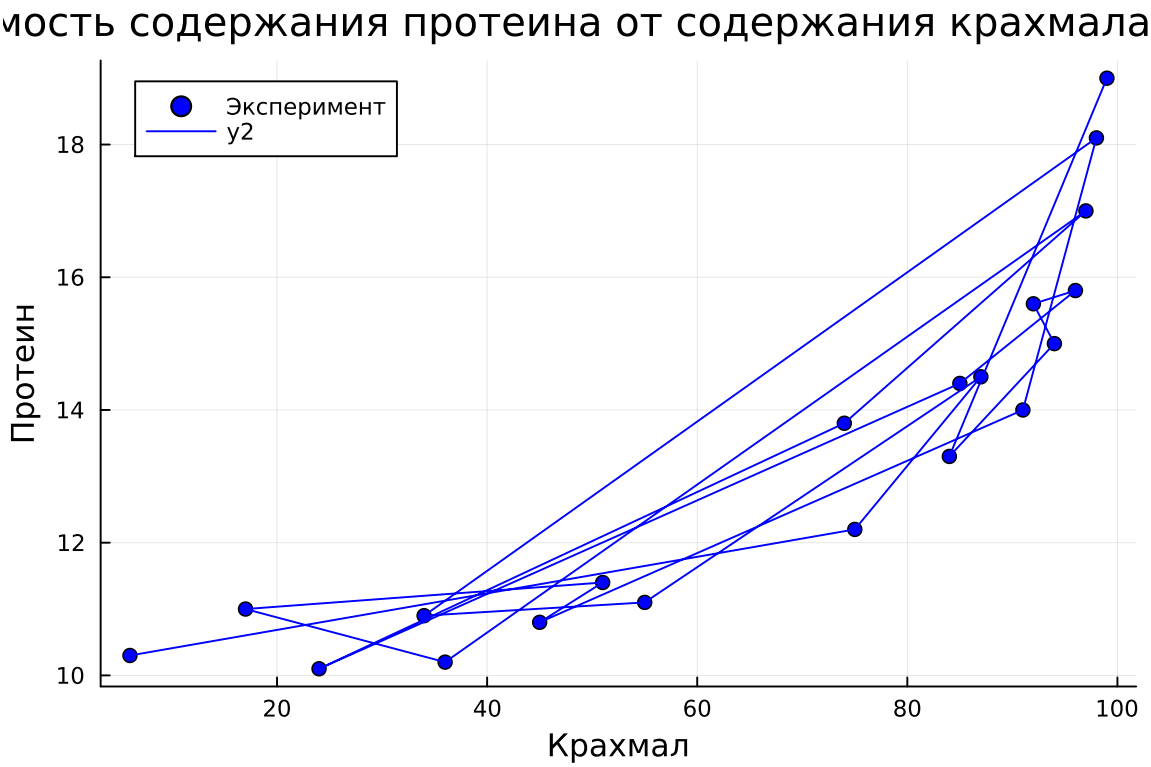
Out[96]:



При попытке соединить точки график ломается

In [97]:
```
using Plots

P = [10.3, 12.2, 14.5, 11.1, 10.9, 18.1, 14.0, 10.8, 11.4, 11, 10.2, 17, 13.
K = [6, 75, 87, 55, 34, 98, 91, 45, 51, 17, 36, 97, 74, 24, 85, 96, 92, 94,
scatter(K, P, label="Эксперимент", color = :blue, title = "Зависимость содер
plot!(K, P, color = :blue)
xlabel!("Крахмал")
ylabel!("Протеин")
```

Out[97]:



Это связано с тем, что массивы P и K не упорядочены по возрастанию значений K. Это может мешать и в других моментах решения задачи, поэтому необходимо упорядочить массивы P и K по возрастанию элементов массива K. В Julia есть функции сортировки, но так как нужна сортировка двух массивов по возрастанию элементов одного из них, напишем это самостоятельно, воспользовавшись алгоритмом «пузырька». Код всей программы решения задачи и результаты её работы приведены ниже.

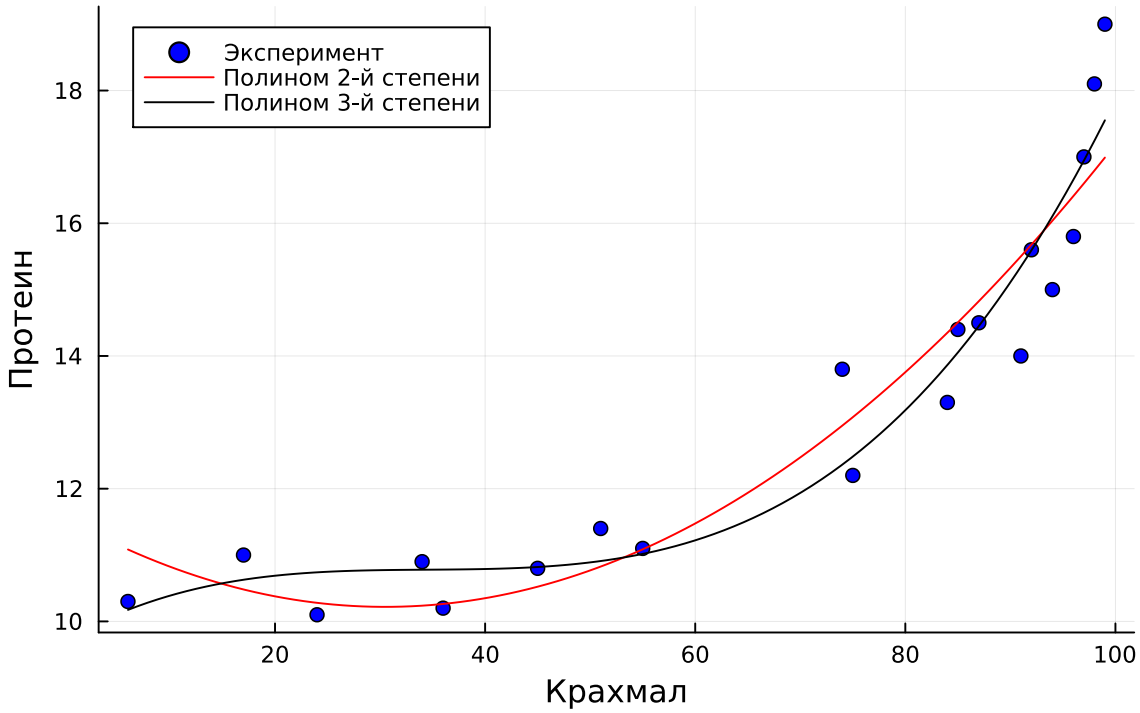
In [98]:

```
using Plots

P = [10.3, 12.2, 14.5, 11.1, 10.9, 18.1, 14.0, 10.8, 11.4, 11, 10.2, 17, 13.
K = [6, 75, 87, 55, 34, 98, 91, 45, 51, 17, 36, 97, 74, 24, 85, 96, 92, 94,
n = length(K)
Mk = sum(K) / n
Mp = sum(P) / n
temp = 0.0

for j in 1:(n - 1)
    for i in 1:(n - 1 - j)
        if K[i] > K[i + 1]
            temp = K[i]
            K[i] = K[i + 1]
            K[i + 1] = temp
            temp = P[i]
            P[i] = P[i + 1]
            P[i + 1] = temp
        end
    end
end
```

```
k = 2
C = ones(k + 1, k + 1)
d = ones(k + 1)

for i in 0:k
    for j in 0:k
        C[i + 1, j + 1] = sum(K .^ (i + j))
    end
    d[i + 1] = sum(P .* K .^ i)
end

a = C \ d

k = 3
C = ones(k + 1, k + 1)
d = ones(k + 1)

for i in 0:k
    for j in 0:k
        C[i + 1, j + 1] = sum(K .^ (i + j))
    end
    d[i + 1] = sum(P .* K .^ i)
end

b = C \ d
println("a= ", a)
println("b= ", b)

t = LinRange(K[1], K[n], 101)

P2 = ones(length(t))
P3 = ones(length(t))

S2 = sum((P .- a[1] .- a[2] .* K .- a[3] .* K .^ 2) .^ 2)
S3 = sum((P .- b[1] .- b[2] .* K .- b[3] .* K .^ 2 .- b[4] .* K .^ 3) .^ 2)
println("Суммарная квадратичная ошибка для полинома 2-й степени= ", S2)
println("Суммарная квадратичная ошибка для полинома 3-й степени= ", S3)
for i in 1:length(t)
    P2[i] = a[1] + a[2] * t[i] + a[3] * t[i] ^ 2
    P3[i] = b[1] + b[2] * t[i] + b[3] * t[i] ^ 2 + b[4] * t[i] ^ 3
end

scatter(K, P, label="Эксперимент", color = :blue)
plot!(t, P2, color = :red, label="Полином 2-й степени")
plot!(t, P3, color = :black, label="Полином 3-й степени")

title!("Зависимость содержания протеина от содержания крахмала в пшенице")
xlabel!("Крахмал")
ylabel!("Протеин")
```

```
a= [11.559505818879044, -0.087941428935383, 0.0014423832660490517]
b= [9.721462632872358, 0.0899538482367657, -0.002579157765581605, 2.4943982962511352e-5]
Суммарная квадратичная ошибка для полинома 2-й степени= 14.09112098649545
Суммарная квадратичная ошибка для полинома 3-й степени= 10.353814397308074
```

Out[98]: мость содержания протеина от содержания крахмала



Задача 3. Линейная интерполяция для температур и растворимости
Давайте решим задачу, в которой нужно найти растворимость вещества в воде при температуре $T = 25^{\circ}C$ на основе данных о растворимости при других температурах.

Исходные данные:

Температура (°C)	Растворимость (г/100 г воды)
0	66.7
5	71
10	76.3
20	85.7
30	92.9
40	99.4
50	113.6

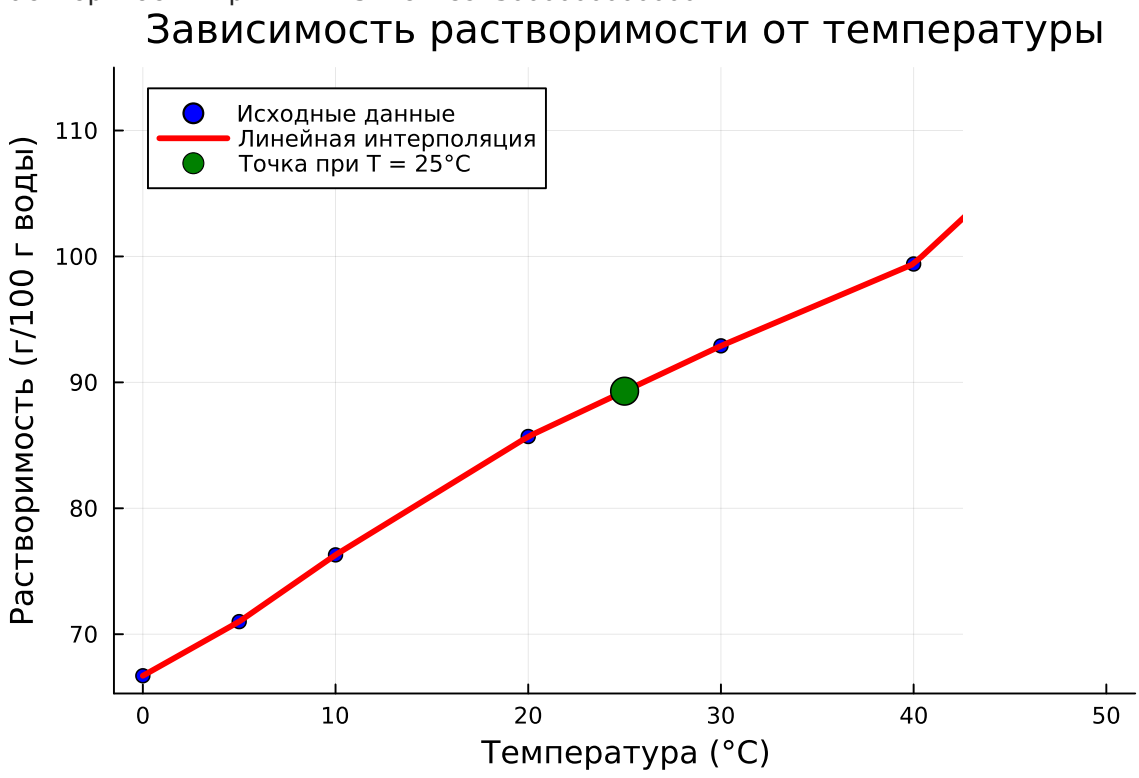
Решим эту задачу при помощи библиотеки **Interpolations.jl**.

In [48]:
```julia
using Interpolations
using Plots

# Данные
T = [0, 5, 10, 20, 30, 40, 50]  # Температуры
P = [66.7, 71, 76.3, 85.7, 92.9, 99.4, 113.6]  # Растворимость
```

```julia
itp = linear_interpolation(T, P)  # Линейная интерполяция

# Интерполяционная функция
f_linear(x) = itp(x)

# Прогноз растворимости при T = 25°C
T_new = 25
P_new = f_linear(T_new)

println("Растворимость при T = $T_new °C: $P_new")

# Создаем новую сетку значений для графика
T_dense = LinRange(0, 50, 100)  # Используем диапазон температур от 0 до 50

# График
scatter(T, P, label="Исходные данные", color=:blue)
plot!(f_linear, T_dense, w=3, label="Линейная интерполяция", color=:red)

# Отметим найденную точку
scatter!([T_new], [P_new], label="Точка при T = 25°C", color=:green, marker=

title!("Зависимость растворимости от температуры")
xlabel!("Температура (°C)")
ylabel!("Растворимость (г/100 г воды)")
```

```
Растворимость при T = 25 °C: 89.30000000000001
```

Out[48]:



Задача 4. Интерполяция для прогнозирования высоты колебаний маятника

Предположим, что вы изучаете движение простого маятника и измеряете его высоту на разных временных интервалах. Данные о высоте маятника были собраны в следующие моменты времени.

Время (секунды)	Высота (м)
1	1.00
2	0.95
3	0.90
4	0.85
5	0.75
6	0.60
7	0.45
8	0.30
9	0.20
10	0.10

Вам необходимо построить модель, которая будет прогнозировать уровень воды в реке в любой момент времени (например, для времени 2.5, 5.5, 8.5 и так далее).

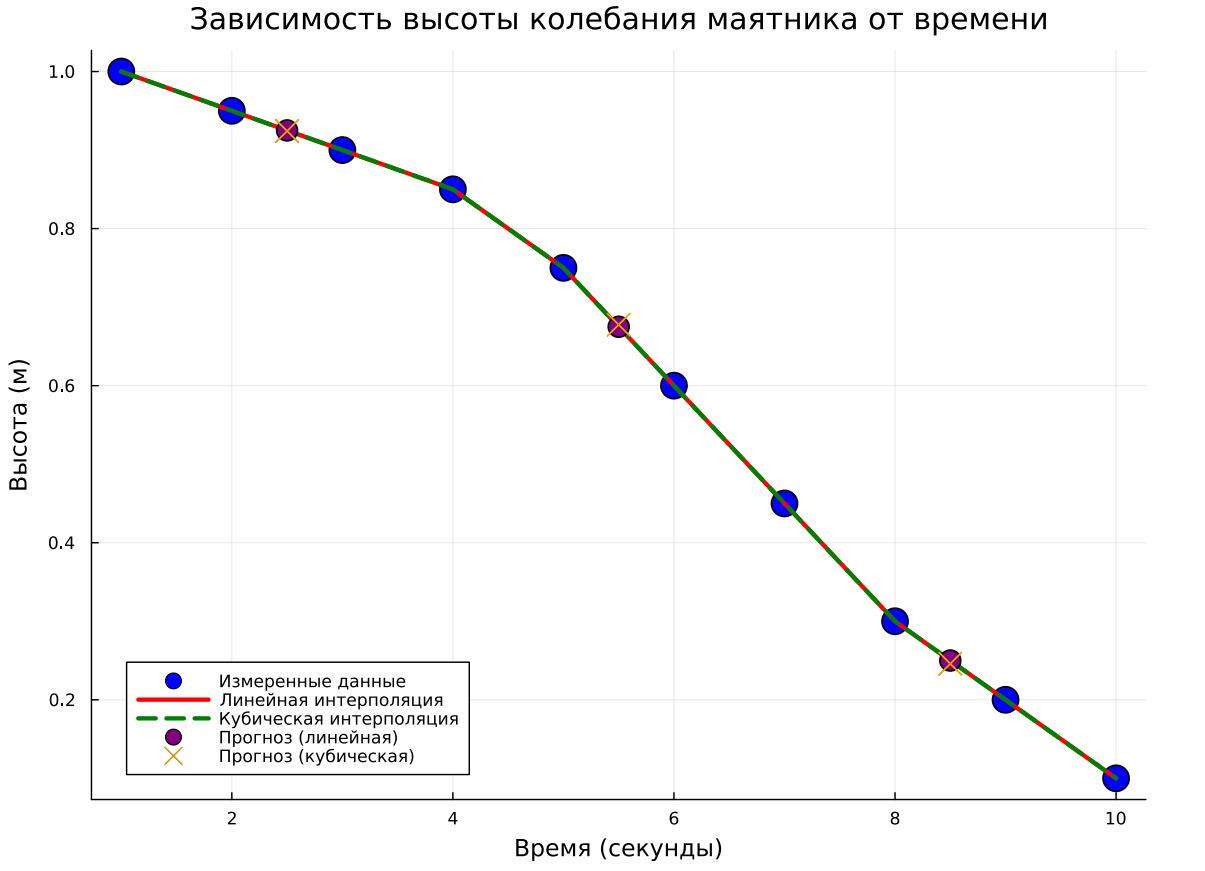
Задание:

1. Используйте линейную и кубическую интерполяцию для прогнозирования высоты маятника между измеренными точками.
2. Постройте график зависимости высоты маятника от времени.
3. Отметьте исходные данные на графике.
4. Прогнозируйте высоту маятника для промежуточных точек времени (например, для времени 2.5, 5.5, 8.5 и других).

In [3]:
```
using Interpolations
using Plots

# Нижняя и верхняя граница интервала
a = 1.0
b = 10.0

time = a:1.0:b  # Время в секундах
height = [1.00, 0.95, 0.90, 0.85, 0.75, 0.60, 0.45, 0.30, 0.20, 0.10]  # Выс

# Интерполяция
itp_linear = linear_interpolation(time, height)
itp_cubic = cubic_spline_interpolation(time, height)  # Кубическая интерполя

# Интерполяционные функции
f_linear(t) = itp_linear(t)
```

```
f_cubic(t) = itp_cubic(t)

# Прогноз для нескольких точек времени (например, 2.5, 5.5, 8.5)
forecast_times = [2.5, 5.5, 8.5]
forecast_linear = [f_linear(t) for t in forecast_times]
forecast_cubic = [f_cubic(t) for t in forecast_times]

# Строим график
scatter(time, height, markersize=10, label="Измеренные данные", color=:blue)

# Строим интерполяционные кривые
plot!(f_linear, time, w=3, label="Линейная интерполяция", color=:red)
plot!(f_cubic, time, linestyle=:dash, w=3, label="Кубическая интерполяция",

# Отметим прогнозируемые точки
scatter!(forecast_times, forecast_linear, label="Прогноз (линейная)", color=
scatter!(forecast_times, forecast_cubic, label="Прогноз (кубическая)", color

# Настройка графика
title!("Зависимость высоты колебания маятника от времени")
xlabel!("Время (секунды)")
ylabel!("Высота (м)")
plot!(size=(800, 600), legend=:bottomleft)
```

Out[3]:



Глава 14. Теория чисел

Теория чисел — это раздел математики, занимающийся изучением свойств целых чисел, их структур и взаимосвязей. Она охватывает такие вопросы, как делимость чисел, разложение на простые множители, алгоритмы нахождения наибольшего общего делителя и наименьшего общего кратного, а также другие теоретико-числовые задачи.

14.1. Теорема деления

Теорема деления (также известная как **формула деления с остатком**)

Теорема утверждает, что для любых двух натуральных чисел a и b ($a \geq b$) существует уникальная пара неотрицательных целых чисел q и r, которые удовлетворяют следующему равенству:

$$a = bq + r,$$

где:

- q — это **целая часть** от деления a на b,
- r — это **остаток** от деления a на b,
- $0 \leq r < b$ — остаток должен быть неотрицательным и строго меньше делителя b.

По другому это можно записать через операцию **mod**.

Операция **mod** — это операция, которая возвращает остаток от деления одного числа на другое. Операция **mod** обозначается как $a \mod b$, где a — делимое, а b — делитель.

$$a \mod b = r$$

где r — это остаток от деления числа a на b.

14.2. Наибольший общий делитель (НОД)

В теории чисел важной задачей является нахождение **наибольшего общего делителя** (НОД) двух чисел. НОД двух чисел — это наибольшее число, которое делит оба исходных числа без остатка.

14.2.1. Алгоритм Евклида

Алгоритм Евклида позволяет находить наибольший общий делитель (НОД) двух чисел a и b с помощью последовательных делений с остатком.

Шаги алгоритма:

1. Пусть a и b — два числа, $a \geq b > 0$.
2. На каждом шаге выполняем деление с остатком $a = bq + r$.
3. Если $r = 0$, то НОД равен b.
4. Иначе, заменяем a на b, a b на r, и повторяем шаги.

In [33]:
```
function euclid(a, b)
    while true
        r = a % b  # Вычисляем остаток от деления
        if r == 0  # Проверяем, равен ли остаток нулю
            return b  # Когда остаток 0, НОД равен последнему ненулевому дел
        else
            # Обновляем значения
            a = b
            b = r
        end
    end
end

# Ввод чисел с клавиатуры
println("Введите первое число (a):")
a = parse(Int, readline())  # Чтение и преобразование в целое число

println("Введите второе число (b):")
b = parse(Int, readline())  # Чтение и преобразование в целое число

# Вызов функции и вывод результата
println("Наибольший общий делитель (НОД) чисел ", a, " и ", b, " равен: ", e
```

```
Введите первое число (a):
Введите второе число (b):
Наибольший общий делитель (НОД) чисел 25 и 4545 равен: 5
```

14.2.2. Расширенный алгоритм Евклида

Алгоритм Евклида, который мы рассмотрели ранее, позволяет находить **наибольший общий делитель (НОД)** двух чисел. Однако, в некоторых задачах нам нужно не только вычислить НОД, но и найти коэффициенты, которые позволяют выразить НОД как линейную комбинацию этих чисел. Именно для этого используется **расширенный алгоритм Евклида**.

Пусть даны два целых числа a и b. Необходимо найти такие целые числа x и y, что:

$$\text{НОД}(a, b) = a \cdot x + b \cdot y,$$

где x и y — целые коэффициенты, называемые **коэффициентами линейной комбинации** для чисел a и b.

Расширенный алгоритм Евклида строится на основе стандартного алгоритма Евклида, но дополнительно отслеживаются коэффициенты x и y.

Мы начинаем с инициализации этих коэффициентов: Изначально для чисел a и b мы устанавливаем $x_0 = 1$, $y_0 = 0$ для a и $x_1 = 0$, $y_1 = 1$ для b.

Далее, на каждом шаге, помимо вычисления нового остатка, также вычисляются новые коэффициенты x_i и y_i .

$$x_{i+1} = x_{i-1} - x_i \cdot q_{i-1}$$
$$y_{i+1} = y_{i-1} - y_i \cdot q_{i-1}$$

Когда остаток становится равным нулю, последний ненулевой остаток будет являться НОД. Коэффициенты x и y в этот момент будут такими, что они удовлетворяют уравнению:

$$\text{НОД}(a, b) = a \cdot x + b \cdot y.$$

In [36]:
```julia
function extended_euclid(a, b)
    x0, x1 = 1, 0  # Начальные значения для x
    y0, y1 = 0, 1  # Начальные значения для y

    while b != 0
        r = a % b  # Вычисляем остаток от деления
        q = a ÷ b  # Вычисляем целую часть от деления

        # Обновляем значения для a и b
        a = b
        b = r

        # Обновляем коэффициенты x и y
        temp_x = x0 - q * x1
        temp_y = y0 - q * y1
        x0, x1 = x1, temp_x
        y0, y1 = y1, temp_y
    end

    return a, x0, y0  # Возвращаем НОД и коэффициенты x и y
end

println("Введите первое число (a):")
a = parse(Int, readline())

println("Введите второе число (b):")
b = parse(Int, readline())

gcd, x, y = extended_euclid(a, b)
println("Наибольший общий делитель (НОД) чисел ", a, " и ", b, " равен: ", g
println("Коэффициенты: x = ", x, ", y = ", y)
```

```
Введите первое число (a):
Введите второе число (b):
Наибольший общий делитель (НОД) чисел 252 и 198 равен: 18
Коэффициенты: x = 4, y = -5
```

14.3. Простые и составные числа

В теории чисел различают два основных класса чисел — простые и составные.

1. **Простое число** — это натуральное число больше единицы, которое делится только на 1 и на себя. То есть, если n — простое число, то его единственными делителями являются 1 и n.

2. **Составное число** — это натуральное число больше единицы, которое не является простым, то есть оно имеет больше двух делителей. Составные числа можно разложить на произведение простых чисел.

14.4.1. Метод пробного деления

Один из самых прямолинейных методов проверки, является **метод пробного деления**. В этом методе мы проверяем, делится ли число n на любое число от 2 до $\sqrt{n}$. Если n делится на одно из таких чисел, оно составное; если нет — число простое.

Почему достаточно проверять делители до $\sqrt{n}$? Это можно объяснить следующим образом:

Пусть n — составное число, то есть оно может быть представлено как произведение двух чисел a и b:

$$n = a \times b$$

Если оба множителя a и b больше $\sqrt{n}$, то их произведение будет больше n, что невозможно, потому что $a \times b = n$. Поэтому хотя бы один из множителей должен быть меньше или равен $\sqrt{n}$. Следовательно, если мы проверяем делители числа n до $\sqrt{n}$, мы гарантированно находим хотя бы один из множителей, если n составное.

In [31]:
```
function is_prime(n)
    # Проверка на простоту для числа n с использованием метода пробного деле
    if n < 2
        return false  # Числа < 2 не являются простыми
    end
    for i in 2:isqrt(n)  # Проверяем делители от 2 до √n
        if n % i == 0  # Если n делится на i, то оно составное
            return false
        end
    end
    return true  # Если делителей не найдено, число простое
end
```

```
# Ввод числа и проверка на простоту
println("Введите число:")
n = parse(Int, readline())

if is_prime(n)
    println("Число $n простое.")
else
    println("Число $n составное.")
end
```

```
Введите число:
Число 25 составное.
```

Хотя метод пробного деления прост и понятен, он имеет значительные недостатки, особенно для больших чисел. Время работы резко увеличивается с ростом числа n, для очень больших чисел этот метод становится слишком медленным и непрактичным.

14.4.2. Решето Эратосфена

Этот алгоритм позволяет найти все простые числа до заданного числа N. Он состоит в поочередном вычеркивании составных чисел, начиная с 2. После завершения алгоритма все оставшиеся числа — простые.

Алгоритм можно описать следующим образом:

1. Запишем все целые числа от 2 до N. $S = 2, 3, 4, \ldots, N$ — исходный список чисел.
2. Пусть переменная p изначально равно 2 — первому простому числу, вычёркиваем из списка все числа от 2p до N, считая шагами по p.
3. Находим первое незачёркнутое число в списке, большее p, и присваиваем это число переменной p.
4. Повторяем шаги 2 и 3, пока значение $p^2 \leq N$).

In [26]:
```
function Eratosthenes(n)
    # Создаём массив для простых чисел
    prime = fill(true, n)  # Все числа считаем простыми (true)
    prime[1] = false  # 1 не является простым числом

    #Для каждого числа p от 2 до √n
    for p in 2:isqrt(n)
        if prime[p] == true
            # Вычеркиваем все кратные j, начиная с j^2
            for k in p^2:p:n
                prime[k] = false
            end
        end
    end

    #Выводим все простые числа
    for p in 2:n
```

```
        if prime[p]
            println(p)
        end
    end
end

println("Введите n:")
n = parse(Int, readline())
Eratosthenes(n)
```

```
Введите n:
2
3
5
7
11
13
```

14.4.3. Тест Ферма

Тест Ферма — это один из самых простых вероятностных алгоритмов для проверки простоты числа. Этот тест основан на **малой теореме Ферма**, которая утверждает, что для любого простого числа p и для любого числа a, которое не делится на p, выполняется следующее условие:

$$a^{p-1} \equiv 1 \pmod{p}$$

Это значит, что если число n простое, то для любого числа a где $1 < a < n - 1$ будет выполняться $a^{n-1} \equiv 1 \pmod{n}$. Если это условие не выполняется для некоторого числа a, то n обязательно составное.

In [29]:
```
# Функция для быстрого возведения в степень по модулю
function modd(base, exp, mod)
    result = BigInt(1)  # Устанавливаем результат равным 1
    base = base % mod  # Вычисляем основание по модулю
    while exp > 0  # Цикл, пока показатель степени больше 0
        if exp % 2 == 1  # Если степень нечетная
            result = (result * base) % mod  # Умножаем результат на основани
        end
        exp ÷= 2  # Делим степень на 2
        base = (base * base) % mod  # Возводим основание в квадрат и находим
    end
    return result
end

# Функция для теста Ферма
function fermat_test(n, iterations=25)
    # Проверка на базовые случаи
    if n < 2
        return false  # Числа < 2 не простые
    end
    if n == 2
        return true  # 2 — простое число
```

```
    end
    if n % 2 == 0
        return false  # Чётные числа больше 2 не простые
    end

    # Выполнение теста Ферма для случайных оснований
    for i in 1:iterations
        a = rand(2:n-2)  # Генерируем случайное a, где 1 < a < n-1
        if modd(a, n-1, n) != 1
            return false  # Если a^(n-1) mod n != 1, то n составное
        end
    end

    return true  # Если тест прошёл все итерации, то число вероятно простое
end

# Ввод числа и проверка на простоту
println("Введите число:")
n = parse(Int, readline())  # Читаем число с консоли

if fermat_test(n)
    println("Число $n вероятно простое.")
else
    println("Число $n составное.")
end
```

```
Введите число:
Число 13 вероятно простое.
```

14.4.4. Тест Миллера-Рабина

Тест Миллера-Рабина — это вероятностный тест, используемый для проверки простоты чисел. Он является псевдопростым тестом, что означает, что он может ошибаться, но вероятность ошибки можно уменьшить путём многократного применения теста.

Основная идея алгоритма:

1. Сначала проверяются базовые случаи: если n меньше или равно 1, оно не простое; если n равно 2 или 3, оно простое; если n чётное и больше 2, то оно составное.

2. Далее представляем число n - 1 как $n - 1 = 2^s \cdot d$, где d нечётно.

3. Выбираем случайное число a, где $2 \leq a \leq n - 2$, и проверяем условие $a^d \mod n$. Если результат равен 1 или n - 1, то n с высокой вероятностью простое.

4. Если условие не выполнено, то вычесляем $a^{2^i \cdot d} \mod n$ для $i = 1, 2, \ldots, r - 1$. Если на каком-то шаге i результат равен n - 1, то

число вероятно простое. Если результат снова становится равным 1, то число n составное.

5. Если число n не проходит проверку для выбранного числа a, оно составное. Повторяя тест с разными основаниями a, можно с высокой вероятностью утверждать, что число простое, если оно прошло все проверки.

In [25]:

```
# Функция для быстрого возведения в степень по модулю
function modd(base, exp, mod)
    result = BigInt(1)  # Устанавливаем результат равным 1
    base = base % mod   # Основание по модулю
    while exp > 0  # Цикл, пока степень больше 0
        if exp % 2 == 1  # Если степень нечётная
            result = (result * base) % mod  # Умножаем на основание и берём
        end
        exp ÷= 2  # Делим степень на 2
        base = (base * base) % mod  # Возводим основание в квадрат по модулю
    end
    return result
end

function is_prime(n)
    iterations = 25  # Количество итераций для теста Миллера-Рабина

    # Проверка на базовые случаи
    if n < 2
        return false  # Числа < 2 не простые
    end
    if n == 2
        return true  # 2 — простое число
    end
    if n % 2 == 0
        return false  # Чётные числа больше 2 не простые
    end

    # Разложение n - 1 = 2^r * d, где d нечётно
    d = n - 1
    r = BigInt(0)
    while d % 2 == 0  # Пока d чётное, делим его на 2
        d ÷= 2
        r += 1
    end

    # Выполнение теста Миллера-Рабина для случайных оснований
    for i in 1:iterations
        a = rand(2:n-2)  # Генерируем случайное a, где 2 <= a <= n-2
        x = modd(a, d, n)  # Вычисляем x = a^d mod n

        # Если x равно 1 или n-1, продолжаем
        if x != 1 && x != n - 1
            composite = true  # Предполагаем, что n составное
            temp_r = r
            while temp_r > 0  # Цикл для проверки степеней x
                x = modd(x, 2, n)  # x = x^2 mod n
```

```
                if x == n - 1
                    composite = false  # n простое, выходим
                    break
                end
                temp_r -= 1
            end

            # Если x не стало равно n-1, число составное
            if composite
                return false  # Число составное
            end
        end
    end

    return true  # Если тест прошёл все итерации, число вероятно простое
end

# Ввод числа и проверка на простоту
println("Введите число:")
n = parse(BigInt, readline())  # Читаем число с консоли

if is_prime(n)
    println("Число $n простое.")
else
    println("Число $n составное.")
end
```

```
Введите число:
Число 851887113014417036499475709157909588274920608419566173084269646346899
простое.
```

Тест Миллера-Рабина особенно полезен для больших чисел, он быстро определяет, является ли число простым. Это значительно ускоряет процесс проверки, делая алгоритм подходящим для чисел, используемых в криптографии и других областях, где важна скорость обработки данных. Несмотря на то, что тест является вероятностным, его использование с достаточным количеством итераций даёт очень высокую вероятность правильного результата, что делает его удобным для практического применения.

14.4. Разложение чисел на множители

Теорема о разложении на множители.

Каждое целое число $n > 1$ можно разложить на произведение простых чисел, то есть существует представление

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k},$$

где $p_1, p_2, \ldots, p_k$ — простые числа, а $a_1, a_2, \ldots, a_k$ — неотрицательные целые числа.

Алгоритм разложения на множители методом пробных делений.

Метод пробных делений (или метод деления с остатком) является одним из самых простых и интуитивно понятных способов разложения числа на простые множители. Суть метода заключается в том, чтобы попытаться разделить число n на все возможные простые числа, начиная с самого маленького, пока не будут найдены все простые множители числа.

Этот метод работает путем последовательного деления числа на простые числа, начиная с 2 и продолжая до тех пор, пока число не станет равным 1. Если при делении на какое-то число остаток от деления равен нулю, то это число является делителем исходного числа, и его нужно записать как множитель.

In [3]:
```
function factorize(n)
    factors = Int[]  # Массив для хранения множителей
    i = 2  # Начинаем с самого маленького простого числа

    while n > 1
        while n % i == 0  # Если число делится на i
            push!(factors, i)  # Добавляем i в список множителей
            n ÷= i  # Делим n на i
        end
        i += 1  # Увеличиваем i и продолжаем искать следующие делители
    end
    return factors
end

println("Введите n:")
n = parse(Int, readline())
println("Множители числа $n: ", factorize(n))
```

```
Введите n:
Множители числа 8: [2, 2, 2]
```

14.5. Функция Эйлера

Функция Эйлера $\varphi(n)$ для целого числа n определяет количество чисел, которые взаимно просты с n , то есть таких чисел, которые не имеют общих делителей с n, кроме 1.

Свойства функции Эйлера

1. **Если n простое**, то $\varphi(n) = n - 1$.

Это объясняется тем, что все числа от 1 до n-1 будут взаимно простыми с n.

2. **Если n составное и разлагается на простые множители**, то функция Эйлера для числа n, разлагаемого на простые множители $p_1, p_2, \ldots, p_k$, вычисляется по формуле:

$$\varphi(n) = n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot \ldots \cdot \left(1 - \frac{1}{p_k}\right)$$

In [18]:
```julia
# Функция для вычисления функции Эйлера с использованием квадратного корня
function euler(n)
    result = n   # Начальное значение для результата

    # Перебор простых чисел от 2 до sqrt(n) с шагом 1
    for p in 2:isqrt(n)
        if n % p == 0  # Если p делит n
            # Уменьшаем результат с учетом p
            while n % p == 0
                n ÷= p   # Делим n на p, пока p делит n
            end
            result -= result ÷ p  # Применяем формулу (1 - 1/p)
        end
    end

    # Если n > 1, то n — это последний простой множитель
    if n > 1
        result -= result ÷ n  # Применяем формулу (1 - 1/n) для последнего r
    end

    return result
end

n = 30
println("Функция Эйлера для числа $n: ", euler(n))
```

```
Функция Эйлера для числа 30: 8
```

Глава 15. Криптография

Основные определения и понятия.

Криптография — это наука и искусство обеспечения безопасности информации. Она охватывает методы шифрования, которые позволяют преобразовать читаемую информацию в недоступный для понимания вид, обеспечивая таким образом защиту данных от несанкционированного доступа.

Шифрование — это процесс преобразования открытого текста (исходной информации) в зашифрованный текст с помощью криптографического алгоритма.

Дешифрование — это процесс восстановления исходной (открытой) информации из зашифрованного текста.

15.1. Пример шифрования с использованием Unicode

Unicode — это стандарт кодирования, который позволяет представлять символы различных языков, включая латиницу, кириллицу, иероглифы и множество других символов. Каждому символу в Unicode соответствует уникальный числовой код (или кодовая точка).

Для начала рассмотрим самый простой способ шифрования, который заключается в том, чтобы преобразовать символы в их код Unicode. В этом методе каждый символ текста будет просто преобразован в его числовое представление, а затем при необходимости обратно в символ. Это будет простая форма шифрования, которая поможет нам понять, как работает кодировка.

In [17]:
```
#Пример шифрования
function string_to_int(message::String)
    result = BigInt(0)           # Инициализируем результат как большое цело
    base = BigInt(65536)         # Основание для кодирования (65536 - 16 бит
    #Для большинства символов в Unicode можно использовать 16-битное предста

    for c in message
        char_code = Int(c)       # Получаем Unicode код символа
        result = result * base + BigInt(char_code)  # Добавляем код символа
    end

    return result
end

message = "Привет"
encoded = string_to_int(message)
println("Шифрование сообщения: ", encoded)
```

```
Шифрование сообщения: 1275436810054983563020141634
```

In [18]:
```
#Пример дешифрования
function int_to_string(num::BigInt)
    result = ""
    base = BigInt(65536)  # Основание для Unicode кодов (16 бит)

    while num > 0
        char_code = Int(num % base)  # Извлекаем код символа
        result = Char(char_code) * result  # Преобразуем код в символ и доба
        num ÷= base  # Разделяем число на основание
    end

    return result
end

# Пример использования
encoded_number = BigInt(1275436810054983563020141634)
```

```
decoded_string = int_to_string(encoded_number)
println("Дешифрование сообщения: ", decoded_string)
```

```
Дешифрование сообщения: Привет
```

Ключ шифрования — это параметр или секретная последовательность символов, используемая для выполнения криптографической операции (шифрования или дешифрования) с целью защиты данных. В криптографии ключ — это значение, которое используется для преобразования исходных данных в зашифрованный (или обратно — из зашифрованного в исходный) вид.

Каждый криптографический алгоритм использует ключи по-разному. Ключ шифрования может быть числом, строкой или даже более сложной структурой данных, в зависимости от типа алгоритма.

Современная криптография делится на два типа: **симметричную и асимметричную**.

Симметричное шифрование — это метод шифрования данных, при котором используется один и тот же ключ для шифрования и дешифрования информации.

Асимметричное шифрование — это метод шифрования, при котором используются два различных ключа: открытый ключ для шифрования и закрытый ключ для дешифрования.

Открытый ключ — это часть ключевой пары, доступная для всех пользователей системы. Он используется для шифрования данных и может быть публично распространён, в отличие от закрытого ключа.

Закрытый ключ — это секретная часть ключевой пары, которая используется для дешифрования данных, зашифрованных с использованием открытого ключа. Закрытый ключ должен храниться в безопасности и не должен передаваться третьим лицам.

Давайте рассмотрим пример, где в качестве ключа для шифрования мы будем использовать сдвиг Unicode. Это значит, что для каждого символа в строке мы будем сдвигать его код (по стандарту Unicode) на некоторое фиксированное количество позиций, которое и будет нашим ключом.

In [21]:
```
# Функция для шифрования сообщения с использованием сдвига Unicode
function shift_encrypt(message::String, key::Int)
    result = BigInt(0)
    base = BigInt(65536)
    for c in message
        # Получаем код символа Unicode
        char_code = Int(c) + key  # Сдвигаем код символа на величину ключа
        result = result * base + BigInt(char_code)  # Преобразуем сдвинутый
```

```
    end
    return result
end

# Пример использования
message = "Привет"  # Исходное сообщение
key = 5  # Ключ сдвига

# Шифруем сообщение
encrypted_message = shift_encrypt(message, key)
println("Зашифрованное сообщение: ", encrypted_message)
```

```
Зашифрованное сообщение: 128148153138818447380014803 9
```

Чтобы расшифровать зашифрованное сообщение, нужно знать ключ — величину сдвига, которую мы использовали при шифровании.

In [23]:
```
#Пример дешифрования
function int_to_string(num::BigInt, key::Int)
    result = ""
    base = BigInt(65536)  # Основание для Unicode кодов (16 бит)

    while num > 0
        char_code = Int(num % base - key)  # Извлекаем код символа
        result = Char(char_code) * result  # Преобразуем код в символ и доба
        num ÷= base  # Разделяем число на основание
    end

    return result
end

# Пример использования
encoded_number = BigInt(128148153138818447380014803 9)
key = 5

decoded_string = int_to_string(encoded_number, key)
println("Дешифрование сообщения: ", decoded_string)
```

```
Дешифрование сообщения: Привет
```

15.2. Шифр Цезаря

Теперь рассмотрим шифр Цезаря, который похож на то, что мы рассматривали ранее с сдвигом Unicode, но в отличие от этого шифр Цезаря применяет сдвиг по буквам в алфавите.

Шифр Цезаря — это классический метод шифрования, в котором каждый символ текста сдвигается на некоторое фиксированное количество позиций в алфавите. Например, если мы используем сдвиг на 3, то буква "А" станет "Г", "Б" станет "Д", и так далее.

In [38]:
```
# Функция для шифрования с использованием шифра Цезаря
function caesar_encrypt(message::String, key::Int)
```

```
    encrypted_message = ""  # Инициализируем пустую строку для хранения заши

    # Мы перебираем каждый символ char в исходном сообщении message
    for char in message
        if islowercase(char) # Проверяется, является ли символ строчной букв
            base = 'a'  # Если символ — строчная буква, то используем 'a' ка
            encrypted_char = Char(mod((Int(char) - Int(base) + key), 32) + I
            #Int(char) - Int(base): Переводим символ в число, вычитая код си
            #(Int(char) - Int(base) + key): Добавляем сдвиг shift к числовом
            #mod(..., 32): Оператор mod используется для того, чтобы после д
            #(например, после буквы 'я') снова вернуться к началу алфавита.
            #+ Int(base): После применения сдвига возвращаемся к коду символ
            encrypted_message *= encrypted_char  # Добавляем зашифрованный с
        elseif isuppercase(char) # Если символ является прописной буквой
            base = 'A'  # Устанавливаем базовый символ для прописного алфави
            # Преобразуем символ в число, сдвигаем его на 'key' позиций,
            # и потом переводим обратно в символ, учитывая ограничение длины
            encrypted_char = Char(mod((Int(char) - Int(base) + key), 32) + I
            encrypted_message *= encrypted_char  # Добавляем зашифрованный с
        else
            # Если символ не является буквой (например, пробел, знак препина
            encrypted_message *= char  # Оставляем его без изменений
        end
    end

    return encrypted_message  # Возвращаем зашифрованное сообщение
end


# Функция для дешифрования с использованием шифра Цезаря
function caesar_decrypt(encrypted_message::String, key::Int)
    return caesar_encrypt(encrypted_message, -key)  # Для дешифрования сдвиг
end

# Пример использования
println("Введите сообщение:")
message = readline()

println("Введите ключ:")
key = parse(Int, readline())

# Шифруем сообщение
encrypted_message = caesar_encrypt(message, key)
println("Зашифрованное сообщение: ", encrypted_message)

# Дешифруем сообщение
decrypted_message = caesar_decrypt(encrypted_message, key)
println("Дешифрование сообщения: ", decrypted_message)
```

```
Введите сообщение:
Введите ключ:
Зашифрованное сообщение: Фхнзкч Снх
Дешифрование сообщения: Привет Мир
```

15.3. Шифр Атбаш

Шифр Атбаш — это один из самых простых шифров, в котором буквы алфавита заменяются на противоположные. Например, 'А' заменяется на 'Я', 'Б' на 'Ю' и так далее.

In [31]:
```julia
# Функция для шифрования с использованием шифра Атбаш для русского и латинск
function atbash_cipher(message::String)
    encrypted_message = ""
    for char in message
        if 'А' <= char <= 'Я'  # Для заглавных русских букв
            base = 'А'
            encrypted_char = Char(Int(base) + (32 - (Int(char) - Int(base))))
            encrypted_message *= encrypted_char
        elseif 'а' <= char <= 'я'  # Для строчных русских букв
            base = 'а'
            encrypted_char = Char(Int(base) + (32 - (Int(char) - Int(base))))
            encrypted_message *= encrypted_char
        else
            encrypted_message *= char  # Если это не буква, просто добавляем
        end
    end
    return encrypted_message  # Возвращаем зашифрованное сообщение
end

# Пример использования
message = "Привет, мир Я!"  # Пример на русском языке
println("Исходное сообщение: ", message)

# Шифруем сообщение
encrypted_message = atbash_cipher(message)
println("Зашифрованное сообщение: ", encrypted_message)

# Дешифруем сообщение (поскольку Атбаш является симметричным шифром)
decrypted_message = atbash_cipher(encrypted_message)
println("Дешифрованное сообщение: ", decrypted_message)
```

```
Исходное сообщение: Привет, мир Я!
Зашифрованное сообщение: Сршюыо, фшр Б!
Дешифрованное сообщение: Привет, мир Я!
```

15.4. Алгоритм XOR-шифрования

XOR-шифрование — это один из самых простых, но в то же время мощных алгоритмов симметричного шифрования. Он использует операцию XOR (исключающее ИЛИ), которая работает с отдельными битами данных.

Основная идея XOR-шифрования заключается в том, что для шифрования и дешифрования используется одна и та же операция — побитовая операция XOR. Благодаря этому алгоритм обладает свойством симметричности: для расшифровки зашифрованного сообщения достаточно снова применить XOR с тем же самым ключом.

Что такое операция XOR?

XOR (исключающее ИЛИ) — это логическая операция, которая действует поразрядно, сравнивая два числа побитово. Суть этой операции заключается в том, что она возвращает **1**, если соответствующие биты двух чисел различны, и **0**, если они одинаковы.

Таблица истинности для операции XOR:

x	y	x ⊻ y
0	1	1
0	0	0
1	1	0
1	0	1

Чтобы выполнить операцию **XOR** в Julia, используется оператор `⊻`. Ниже приведены примеры использования этой операции.

In [1]:
```julia
# Функция для выполнения XOR и вывода результата в двоичном формате
function xor_binary(x::UInt8, y::UInt8)
    # Выполнение операции XOR
    result = x ⊻ y

    # Выводим результат в двоичной строке
    println("Число x в двоичной системе: ", bitstring(x)) #функцию bitstring
    println("Число y в двоичной системе: ", bitstring(y))
    println("Результат XOR в двоичной системе: ", bitstring(result))

    return result
end

# Пример с числами
x = 0b10101010  # 0b — это префикс, который говорит компилятору или интерпре
y = 0b11001100  #

# Выполнение XOR и вывод результатов
xor_binary(x, y)
```

```
Число x в двоичной системе: 10101010
Число y в двоичной системе: 11001100
Результат XOR в двоичной системе: 01100110
```

Out[1]: 0x66

Основные шаги XOR-шифрования

1. **Преобразование сообщения в байты**: Сообщение (строка) представляется в виде последовательности байтов. Каждый символ строки преобразуется в его числовое представление

2. **Подготовка ключа**: Ключ также преобразуется в числовое представление. Если длина ключа меньше длины сообщения, ключ расширяется (например, повторяется несколько раз), чтобы его длина соответствовала длине сообщения.

3. **Применение операции XOR**: Для каждого символа сообщения применяется операция XOR.

4. **Получение зашифрованного сообщения**: Результатом XOR-операции является зашифрованный текст, который состоит из зашифрованных байтов. Эти байты могут быть преобразованы обратно в строку для отображения зашифрованного сообщения.

5. **Дешифровка**: Для расшифровки сообщения нужно снова применить ту же операцию XOR с тем же ключом. Поскольку операция XOR является обратимой, зашифрованное сообщение будет расшифровано обратно в исходное.

In [4]:
```
# Функция для XOR-шифрования и дешифрования
function xor_encrypt_decrypt(message::String, key::String)
    # Преобразуем ключ в байтовое представление (массив чисел)
    key_bytes = [Int(c) for c in key]
    message_bytes = [Int(c) for c in message]

    key_extended = repeat(key_bytes, outer = ceil(Int, length(message) / ler
    #repeat(key_bytes, outer = ...) повторяет массив key_bytes, создавая нов
    #который содержит несколько копий оригинального массива ключа
    #Аргумент outer = ceil(Int, length(message) / length(key_bytes)) рассчит
    #чтобы его длина стала хотя бы равной длине сообщения. ceil(Int, ...) ок
    key_extended = key_extended[1:length(message)]
    #key_extended = key_extended[1:length(message)] обрезает расширенный мас
    #чтобы длина key_extended точно совпала с длиной сообщения.

    # Применяем операцию XOR к каждому символу
    encrypted_message = Char[Char(message_bytes[i] ⊻ key_extended[i]) for i

    return String(encrypted_message)  # Преобразуем массив Char обратно в ст
end

# Пример использования
println("Введите сообщение:")
message = readline()
println("Введите ключ:")
key = readline()

# Шифруем сообщение
encrypted_message = xor_encrypt_decrypt(message, key)
println("Зашифрованное сообщение: ", encrypted_message)

# Дешифруем сообщение (поскольку XOR работает одинаково для шифрования и деш
```

```
decrypted_message = xor_encrypt_decrypt(encrypted_message, key)
println("Дешифрованное сообщение: ", decrypted_message)
```

```
Введите сообщение:
Введите ключ:
Зашифрованное сообщение: ХЎIЋvЙў Љ□mzg□ЄЙψvЋЇSJЙO
Дешифрованное сообщение: Алгоритм XOR-шифрования
```

15.5. Алгоритм RSA

Алгоритм **RSA** (по имени авторов **Rivest**, **Shamir**, и **Adleman**) является одним из самых популярных и широко используемых асимметричных алгоритмов шифрования. В отличие от симметричных алгоритмов, которые используют один и тот же ключ для шифрования и дешифрования, RSA использует пару ключей: **открытый** и **закрытый**.

Принцип работы RSA

1. **Генерация ключей**

   - Выбираются два больших простых числа p и q (для выбора подходящих простых чиселчасто используется тест Миллера-Рабина подробнее описанный в п.12.4.4.).
   - Вычисляется их произведение $r = p \times q$, которое используется для формирования открытого и закрытого ключей.
   - Вычисляется функция Эйлера $\varphi(r) = (p-1) \cdot (q-1)$.
   - Открытый ключ e выбирается таким образом, чтобы он был взаимно прост с $\varphi(r)$ (то есть, $\gcd(e, \varphi(r)) = 1$) (для нахождения e используем стандартный алгоритм Евклида п. 12.2.1.)
   - Закрытый ключ d должно удовлетворять условию $d \times e \equiv 1 \pmod{\varphi(r)}$ (для нахождения d можно использовать расширенный алгоритм Евклида п. 12.2.2.).
2. **Шифрование**

   - Для шифрования используется открытый ключ (e, r).
   - Сообщение M преобразуется в число m, которое должно быть меньше r.
   - Шифрование выполняется по формуле:

   $$c = m^e \pmod{n},$$

   - где c — это зашифрованное сообщение.
3. **Дешифрование**

   - Для дешифрования используется закрытый ключ (d, r).
   - Дешифрование выполняется по формуле:

$$m = c^d \pmod{n}$$

- Полученное число m затем можно преобразовать обратно в сообщение.

In [6]:

```
using Random
using Dates

# Функция для быстрого возведения в степень по модулю
function modd(base, exp, mod)
    result = BigInt(1)
    base = base % mod
    while exp > 0
        if exp % 2 == 1
            result = (result * base) % mod
        end
        exp ÷= 2
        base = (base * base) % mod
    end
    return result
end

# Тест Миллера-Рабина
function is_prime(n)
    iterations = 25
    if n < 2
        return false  # Числа < 2 не простые
    end
    if n == 2
        return true  # 2 — простое число
    end
    if n % 2 == 0
        return false  # Чётные числа больше 2 не простые
    end

    d = n - 1
    r = BigInt(0)
    # Найти d такое, что n - 1 = 2^r * d
    while d % 2 == 0# Пока d четное, делим d на 2 и увеличиваем r
        d ÷= 2
        r += 1
    end

    for i in 1:iterations
        a = rand(2:n-2)  # Генерируем случайное a < n - 1
        x = modd(a, d, n)  # x = a^d mod n
        if x != 1 && x != n - 1# Проверка, не равен ли x 1 или n-1
            composite = true # Предполагаем, что n составное
            temp_r = r
            while temp_r > 0 # Цикл для проверки значений x
                x = modd(x, BigInt(2), n)  # x = x^2 mod n
                if x == n - 1
                    composite = false # n простое
                    break # Выходим из цикла
                end
                temp_r -= 1
```

```
            end

        # Если x не стало равно n-1, число составное
            if composite
                return false
            end
        end
    end
    return true
end


function string_to_int(message::String)
    result = BigInt(0)
    base = BigInt(65536)  # 65536 для Unicode кодов (с использованием 16 бит

    for c in message
        # Получаем Unicode код символа
        char_code = Int(c)
        result = result * base + BigInt(char_code)  # Добавляем код символа
    end
    return result
end

function int_to_string(num::BigInt)
    result = ""
    base = BigInt(65536)

    while num > 0
        char_code = Int(num % base)  # Получаем код символа
        result = Char(char_code) * result  # Преобразуем в символ и добавляе
        num ÷= base  # Делим число на основание 65536
    end

    return result
end


# Алгоритм Евклида
function euclid(a, b)
    r_0 = a
    r_1 = b
    while true
        r_2 = r_0 % r_1
        if r_2 == 0
            if r_1 == 1
                return true
            else
                return false
            end
        else
            r_0 = r_1
            r_1 = r_2
        end
    end
end
```

```
# Расширенный алгоритм Евклида
function advanced_euclid(a, b)
    r_0 = a
    r_1 = b
    x_0 = 1
    x_1 = 0
    while true
        r_2 = r_0 % r_1
        q = r_0 ÷ r_1
        x_2 = x_0 - q * x_1
        if r_2 == 0
            if r_1 == 1
                return x_1
            end
        else
            r_0 = r_1
            r_1 = r_2
            x_0 = x_1
            x_1 = x_2
        end
    end
end

# Функция для генерации случайного простого числа длины n
function generate_random(length)
    while true
        startt = BigInt(10)^(length-1)
        endd = BigInt(10)^length - 1
        n = rand(startt:endd)
        if is_prime(n)
            return n
        end
    end
end

# Открытый ключ
function public_key(phi_r, length)
    while true
        startt = BigInt(10)^(length-1)
        endd = BigInt(10)^length - 1
        e = rand(startt:endd)
        if euclid(e, phi_r)
            return e
        end
    end
end

# Закрытый ключ
function private_key(e, phi_r)
    d = advanced_euclid(e, phi_r)
    if d < 0
        d = d + phi_r
    end
    return d
end
```

```
# Пример использования
start_time = now()

length_p = rand(20:40) #длина простого числа p
length_q = rand(20:40) #длина простого числа q
length_e = rand(20:40) #длина ключа e


println("Введите текст:")
text = readline()

# Генерация простых чисел p и q
p = generate_random(length_p)
q = generate_random(length_q)
r = p * q
phi_r = (p - 1) * (q - 1)

# Генерация открытого и закрытого ключей
e = public_key(phi_r, length_e)
d = private_key(e, phi_r)

# Преобразование текста в число
message_int = string_to_int(text)

# Шифрование
m_1 = modd(message_int, e, r)

# Дешифрование
m_2 = modd(m_1, d, r)

# Преобразование числа обратно в строку
decrypted_message = int_to_string(m_2)

# Печать результатов
end_time = now()
elapsed_time = end_time - start_time

println("Сгенерированное простое число p: $p")
println("Сгенерированное простое число q: $q")
println("Перемножение r: $r")
println("Функция Эйлера φ(r): $phi_r")
println("Открытый ключ e: $e")
println("Закрытый ключ d: $d")
println("Текст: $text")
println("Шифрование: $m_1")
println("Дешифрование: $m_2")
println("Дешифрованный текст: $decrypted_message")
println("Время выполнения: $elapsed_time")
```

```
Введите текст:
```

```
Сгенерированное простое число p: 221219417314810518551335135956767
Сгенерированное простое число q: 997487954390753901984795992325277 9
Перемножение r: 220663704048864868558033811003195577408238797996241935747809
56605493
Функция Эйлера φ(r): 220663704048864868558033811003195455537501627439799882 13285897395948
Открытый ключ e: 44528747329680538180792334297622624030 1
Закрытый ключ d: 201891725194242700479230327954946131188643847991956897451854
20279641
Текст: Я тебя люблю
Шифрование: 994719057602734114378626618363318697213941592307315312255040262 3752
Дешифрование: 10258146705907100880133421164069737975451417527106004692 6
Дешифрованный текст: Я тебя люблю
Время выполнения: 857 milliseconds
```

Глава 16. Численное решение обыкновенных дифференциальных уравнений

При решении задач математического моделирования очень часто приходится решать обыкновенные дифференциальные уравнения.

Напомним численные методы решения дифференциального уравнения первого порядка. Будем рассматривать для следующей *задачи Коши*. Найти решение дифференциального уравнения

$$x' = f(x,t),$$

удовлетворяющего начальному условию

$$x(t_0) = x_0$$

иными словами, требуется найти интегральную кривую $x = x(t)$, проходящую через заданную точку $M_0(t_0, x_0)$.

Для дифференциального уравнения n-го порядка

$$x^{(n)} = f(t, x, x^{'}, x^{''}, \ldots, x^{(n-1)})$$

задача Коши состоит в нахождении решения $x = x(t)$, удовлетворяющего уравнению и начальным условиям

$$x(t_0) = x_0, x^{'}(t_0) = x_0^{'}, \ldots, x^{(n-1)}(t_0) = x_0^{(n-1)}$$

Рассмотрим основные численные методы решения задачи Коши.

16.1. Решение дифференциальных уравнений методом Эйлера

При решении задачи Коши на интервале $[t_0, t_n]$, выбрав достаточно малый шаг *h*, построим систему точек

$$t_i = t_0 + ih, \quad i = 0, 1, \ldots, n, \quad h = \frac{t_n - t_0}{n}$$

Основная расчётная формула.

$$x_{i+1} = x_i + hf(x_i, t_i), \quad i = 0, 1, \ldots, n-1.$$

16.2. Решение дифференциальных уравнений при помощи модифицированного метода Эйлера

Более точным методом решения задачи Коши является *модифицированный метод Эйлера*, при котором сначала вычисляют промежуточные значения.

$$t_p = t_i + \frac{h}{2},\ x_p = x_i + \frac{h}{2} f(x_i, t_i),$$

после чего находят значение x_{i+1} по формуле

$$x_{i+1} = x_i + hf(x_p, t_p), \quad i = 0, 1, \ldots, n-1$$

16.3. Решение дифференциальных уравнений методами Рунге-Кутта

Рассмотренные выше методы Эйлера (как обычный, так и модифицированный) являются частными случаями явного *метода Рунге-Кутта* k-го порядка. В общем случае формула вычисления очередного приближения методом Рунге-Кутта имеет вид:

$$x_{i+1} = x_i + h\varphi(t_i, x_i, h), \quad i = 0, 1, \ldots, n-1$$

Метод Эйлера является \emph{методом Рунге-Кутта первого порядка} ($k = 1$) и получается при $\varphi(t, x, h) = f(t, x)$.

Семейство *методов Рунге-Кутта* второго порядка имеет вид

$$x_{i+1} = x_i + h\left((1-\alpha) f(t_i, x_i) + \alpha f\left(t_i + \frac{h}{2\alpha}, x_i + \frac{h}{2\alpha} f(t_i, x_i)\right)\right),$$
$$i = 0, 1, \ldots, n-1$$

Два наиболее известных среди методов Рунге-Кутта второго порядка -- это метод Хойна ($\alpha = \frac{1}{2}$) и модифицированный метод Эйлера ($\alpha = 1$). При $\alpha = \frac{1}{2}$ получаем расчётную формулу *метода Хойна*:

$$x_{i+1} = x_i + \frac{h}{2}\left(f(t_i, x_i) + f(t_i + h, x_i + hf(t_i, x_i))\right), \quad i = 0, 1, \ldots, n-1$$

При $\alpha = 1$ получаем расчётную формулу уже рассмотренного выше модифицированного метода Эйлера

$$x_{i+1} = x_i + hf\left(t_i + \frac{h}{2}, x_i + \frac{h}{2}f(t_i, x_i)\right), \quad i = 0, 1, \ldots, n-1$$

Наиболее известным является *метод Рунге-Кутта четвёртого порядка*, расчётные формулы которого можно записать в виде :

$$\begin{cases} x_{i+1} = x_i + \Delta x_i, \quad i = 0, 1, \ldots, n-1 \\ \Delta x_i = \frac{h}{6}(K_1^i + 2K_2^i + 2K_3^i + K_4^i) \\ K_1^i = f(t_i, x_i) \\ K_2^i = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_1^i) \\ K_3^i = f(t_i + \frac{h}{2}, x_i + \frac{h}{2}K_2^i) \\ K_4^i = f(t_i + h, x_i + hK_3^i) \end{cases}$$

Рассмотренные методы Рунге-Кутта относятся к классу *одношаговых методов*, в которых для вычисления значения в очередной точке x_{k+1} нужно знать значение в предыдущей точке x_k.

Ещё один класс методов решения задачи Коши --- *многошаговые методы*, в которых используются точки x_{k-3}, x_{k-2}, x_{k-1}, x_k для вычисления x_{k+1}. В многошаговых методах первые четыре начальные точки (t_0, x_0), (t_1, x_1), (t_2, x_2), (t_3, x_3) должны быть получены заранее любым из одношаговых методов (метод **Эйлера**, **Рунге-Кутта** и т.д.). Наиболее известными многошаговыми методами являются методы прогноза-коррекции **Адамса** и **Милна**.

16.4. Решение дифференциальных уравнений методом Адамса

Рассмотрим решение задачи Коши методом **Адамса**, будем численно решать дифференциальное на интервале $[t_i, t_{i+1}]$, считая, что решение в точках $t_0, t_1, t_2, \ldots, t_i$ уже найдено (метод Эйлера или Рунге-Кутта), и

значения в этих точках будем использовать для нахождения значения $x(t_{i+1})$.

Первое приближение (прогноз) $\tilde{x}_{i+1}$ вычисляется по формуле.

$$\tilde{x}_{i+1} = x_i + \frac{h}{24}(-9f(t_{i-3}, x_{i-3}) + 37f(t_{i-2}, x_{i-2}) - \\ - 59f(t_{i-1}, x_{i-1}) + 55f(t_i, x_i))$$

Как только $\tilde{x}_{i+1}$ вычислено, его можно использовать для вычисления второго приближения (корректор)

$$x_{i+1} = x_i + \frac{h}{24}(f(t_{i-2}, x_{i-2}) - 5f(t_{i-1}, x_{i-1}) + \\ + 19f(t_i, x_i) + 9f(t_{i+1}, \tilde{x}_{i+1}))$$

Таким образом, для вычисления значения $x(t_{i+1})$ методом **Адамса** необходимо последовательно применять формулы для вычисления прогноза $\tilde{x}_{i+1}$ и корректора x_{i+1}, а первые четыре точки можно получить одноточечными методами **Эйлера** или **Рунге-Кутта**.

16.5. Решение дифференциальных уравнений методом Милна

Рассмотрим решение задачи Коши методом **Милна**, будем численно решать дифференциальное на интервале $[t_k, t_{k+1}]$, считая, что решение в точках $t_0, t_1, t_2, \ldots, t_k$ уже найдено (метод Эйлера или Рунге-Кутта), и значения в этих точках будем использовать для нахождения значения $x(t_{k+1})$. Вычислительная схема метода **Милна** состоит в следующем.

Вычисляем первое приближение - ***прогноз Милна*** $\tilde{x}_{k+1}$ для значения функции в точке t_{k+1}

$$\tilde{x}_{k+1} = x_{k-3} + \frac{4h}{3}(2f(t_{k-2}, x_{k-2}) - f(t_{k-1}, x_{k-1}) + 2f(t_k, x_k))$$

Далее вычисляем второе приближение - ***корректор Милна***

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, \tilde{x}_{k+1}))$$

В методе Милна для вычисления значения $x(t_{k+1})$ необходимо последовательно применять формулы для вычисления прогноза $\tilde{x}_{k+1} * *$ и корректора x_{k+1}, а первые четыре точки можно получить методом Рунге-Кутта.

Существует **модифицированный метод Милна**. В нём сначала вычисляется первое приближение - ***прогноз Милна*** $\tilde{x}_{k+1}$, затем вычисляется *управляющий параметр* m_{k+1}

$$m_{k+1} = \tilde{x}_{k+1} + \frac{28}{29}(x_k - \tilde{x}_k),$$

после чего вычисляется значение второго приближения - ***корректор Милна*** по формуле

$$x_{k+1} = x_{k-1} + \frac{h}{3}(f(t_{k-1}, x_{k-1}) + 4f(t_k, x_k) + f(t_{k+1}, m_{k+1}))$$

В модифицированном методе Милна первые четыре точки также можно получить методом **Рунге-Кутта**.

Рассмотрим примеры решения численного обыкновенных дифференциальных уравнений (задач Коши) с помощью методов **Эйлера**, **Рунге-Кутта**, **Милна** и **Адамса**, используя язык программирования *julia*.

Задача 1.

$$y' = y \cdot \frac{x+2}{x+1}, y(0) = 1$$

Точное решение задачи имеет вид $y = (x+1) \cdot e^x$. Ниже приведена программа на языке программирования *Julia* решения этой задачи методами **Эйлера**, **Рунге-Кутта**, **Милна** и **Адамса**.

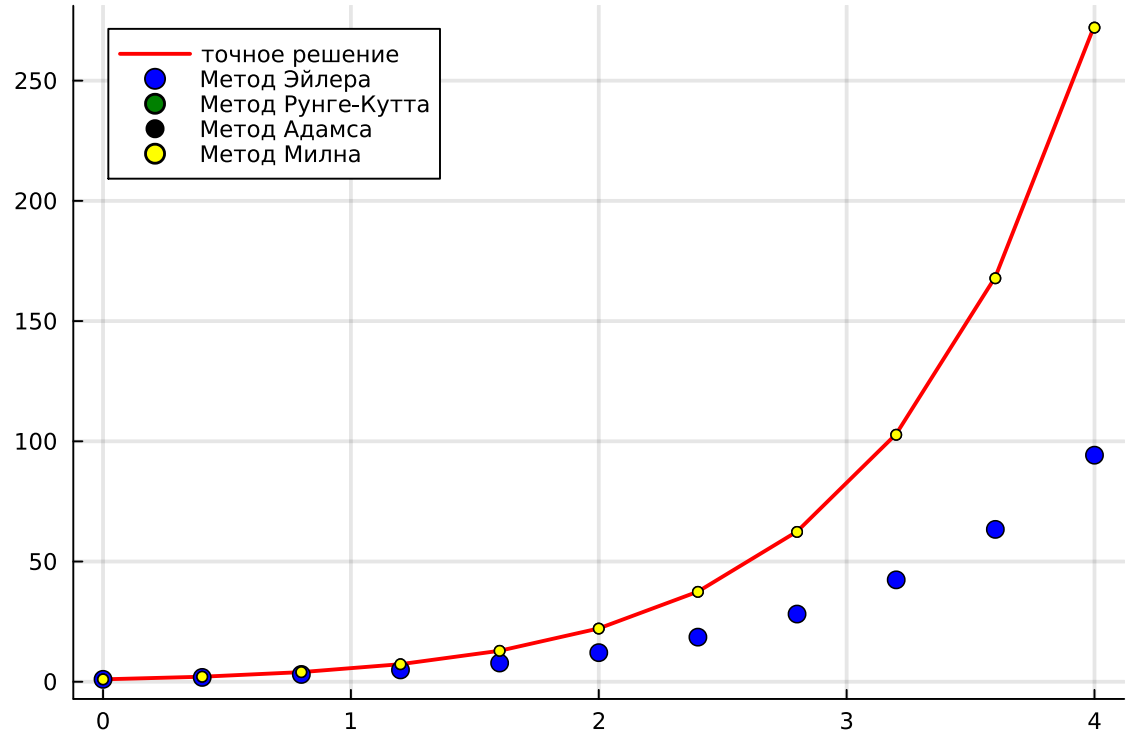
In [3]:
```
using Printf,Plots
# Правая часть диференциального уравнения
f(x,y)=y*(x+2)/(x+1)
# Начальное условие
x0=Float64(0)
y0=Float64(1)
# Количество участков на интервале интегрирования
n=Int64(10)
# Правая граница интервала интегрирования
xn=Float64(4)
# Шаг интегрирования
h=(xn-x0)/n
# Массив абсцисс
x=zeros(n+1)
# Массив для хранения точного решениия
yt=zeros(n+1)
# Цикл для формирования точного решения
for i=1:n+1
    x[i]=x0+(i-1)*h
    yt[i]=(x[i]+1)*exp(x[i])
end
```

```
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(n+1)
ye[1]=y0
# Цикл для формирования решения методом Эйлера
for i=2:n+1
    ye[i]=ye[i-1]+h*f(x[i-1],ye[i-1])
end
# Реализация метода Рунге-Кутта
# Массив для хранения решения методом Рунге-Кутта
yrk=zeros(n+1)
yrk[1]=y0
# Цикл для формирования решения методом Рунге-Кутта
for i=2:n+1
    K1=f(x[i-1],yrk[i-1]);
    K2=f(x[i-1]+h/2,yrk[i-1]+h/2*K1);
    K3=f(x[i-1]+h/2,yrk[i-1]+h/2*K2);
    K4=f(x[i-1]+h,yrk[i-1]+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    yrk[i]=yrk[i-1]+delt;
 end
# Реализация метода Адамса
# Массив для хранения решения методом Адамса
ya=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ya[i]=yrk[i]
end
# Цикл для формирования решения методом Адамса
for i=5:n+1
    yp=ya[i-1]+h/24*(-9*f(x[i-4],ya[i-4])+37*f(x[i-3],ya[i-3])-59*f(x[i-2],y
    ya[i]=ya[i-1]+h/24*(f(x[i-3],ya[i-3])-5*f(x[i-2],ya[i-2])+19*f(x[i-1],ya
end
# Реализация метода Милна
# Массив для хранения решения методом Милна
ym=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ym[i]=yrk[i]
end
# Цикл для формирования решения методом Милна
for i=5:n+1
    yp=ym[i-4]+4*h/3*(2*f(x[i-3],ym[i-3])-f(x[i-2],ym[i-2])+2*f(x[i-1],ym[i-
    ym[i]=ym[i-2]+h/3*(f(x[i-2],ym[i-2])+4*f(x[i-1],ym[i-1])+f(x[i],yp));
end
# Цикл для вывода результатов
for i=1:n+1
    @printf("x=%8.2f, ye=%8.4f, yrk=%8.4f, ya=%8.4f, ym=%8.4f, yt=%8.4f\n",
end
```

```
x=    0.00, ye=  1.0000, yrk=  1.0000, ya=  1.0000, ym=  1.0000, yt=  1.0000
x=    0.40, ye=  1.8000, yrk=  2.0866, ya=  2.0866, ym=  2.0866, yt=  2.0886
x=    0.80, ye=  3.0343, yrk=  4.0000, ya=  4.0000, ym=  4.0000, yt=  4.0060
x=    1.20, ye=  4.9223, yrk=  7.2905, ya=  7.2905, ym=  7.2905, yt=  7.3043
x=    1.60, ye=  7.7862, yrk= 12.8498, ya= 12.8478, ym= 12.8499, yt= 12.8779
x=    2.00, ye= 12.0985, yrk= 22.1132, ya= 22.1066, ym= 22.1129, yt= 22.1672
x=    2.40, ye= 18.5510, yrk= 37.3796, ya= 37.3635, ym= 37.3781, yt= 37.4788
x=    2.80, ye= 28.1539, yrk= 62.3124, ya= 62.2784, ym= 62.3081, yt= 62.4897
x=    3.20, ye= 42.3790, yrk=102.7267, ya=102.6597, ym=102.7166, yt=103.0366
x=    3.60, ye= 63.3668, yrk=167.8190, ya=167.6932, ym=167.7974, yt=168.3519
x=    4.00, ye= 94.2236, yrk=272.0866, ya=271.8580, ym=272.0434, yt=272.9908
```

In [7]:
```
# Графическое решение задачи
Plots.plot(x, yt, color = :red, linewidth = 2, label = "точное решение",gric
Plots.scatter!(x, ye, color = :blue, markersize = 5, label = "Метод Эйлера")
Plots.scatter!(x, yrk, color = :green, markersize = 3, label = "Метод Рунге-
Plots.scatter!(x, ya, color = :black, markersize = 3, label = "Метод Адамса"
Plots.scatter!(x, ym, color = :yellow, markersize = 3, label = "Метод Милна"
```

Out[7]:



Задача 2.

$$y' = y, y(0) = 1.$$

Точное решение этого уравнение имеет вид $y = e^x$. Решая уравнение численно методами **Эйлера**, **Рунге-Кутта**, **Милна** и **Адамса** в точке x=1 должны получить число $e = 2.7182818...$. Решение приведено ниже.

In [8]:
```
# Правая часть диференциального уравнения
f(x,y)=y
# Начальное условие
x0=Float64(0)
```

```
y0=Float64(1)
# Количество участков на интервале интегрирования
n=Int64(10)
# Правая граница интервала интегрирования
xn=Float64(1)
# Шаг интегрирования
h=(xn-x0)/n
# Массив абсцисс
x=zeros(n+1)
# Массив для хранения точного решениия
yt=zeros(n+1)
# Цикл для формирования точного решения
for i=1:n+1
    x[i]=x0+(i-1)*h
    yt[i]=exp(x[i])
end
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(n+1)
ye[1]=y0
# Цикл для формирования решения методом Эйлера
for i=2:n+1
    ye[i]=ye[i-1]+h*f(x[i-1],ye[i-1])
end
# Реализация метода Рунге-Кутта
# Массив для хранения решения методом Рунге-Кутта
yrk=zeros(n+1)
yrk[1]=y0
# Цикл для формирования решения методом Рунге-Кутта
for i=2:n+1
    K1=f(x[i-1],yrk[i-1]);
    K2=f(x[i-1]+h/2,yrk[i-1]+h/2*K1);
    K3=f(x[i-1]+h/2,yrk[i-1]+h/2*K2);
    K4=f(x[i-1]+h,yrk[i-1]+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    yrk[i]=yrk[i-1]+delt;
 end
# Реализация метода Адамса
# Массив для хранения решения методом Адамса
ya=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ya[i]=yrk[i]
end
# Цикл для формирования решения методом Адамса
for i=5:n+1
    yp=ya[i-1]+h/24*(-9*f(x[i-4],ya[i-4])+37*f(x[i-3],ya[i-3])-59*f(x[i-2],y
    ya[i]=ya[i-1]+h/24*(f(x[i-3],ya[i-3])-5*f(x[i-2],ya[i-2])+19*f(x[i-1],ya
end
# Реализация метода Милна
# Массив для хранения решения методом Милна
ym=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ym[i]=yrk[i]
end
```

```
# Цикл для формирования решения методом Милна
for i=5:n+1
    yp=ym[i-4]+4*h/3*(2*f(x[i-3],ym[i-3])-f(x[i-2],ym[i-2])+2*f(x[i-1],ym[i-
    ym[i]=ym[i-2]+h/3*(f(x[i-2],ym[i-2])+4*f(x[i-1],ym[i-1])+f(x[i],yp));
end
# Вывод числа e, которое получено путем численного решения оду y'=y разными
    @printf("x=%8.2f, ye=%8.6f, yrk=%8.6f, ya=%8.6f, ym=%8.6f, yt=%8.6f\n",
```

```
x=    1.00, ye=2.593742, yrk=2.718280, ya=2.718284, ym=2.718282, yt=2.718282
```

Задача 3.

$$y' = \frac{4}{1+x^2}, y(0) = 0.$$

Точное решение этого уравнение имеет вид $y = arctgx$. Решая уравнение численно методами **Эйлера**, **Рунге-Кутта**, **Милна** и **Адамса** в точке x=1 должны получить число $\pi = 3.14159....$ Решение приведено ниже.

In [9]:

```
# Правая часть диференциального уравнения
f(x,y)=4/(1+x*x)
# Начальное условие
x0=Float64(0)
y0=Float64(0)
# Количество участков на интервале интегрирования
n=Int64(10)
# Правая граница интервала интегрирования
xn=Float64(1)
# Шаг интегрирования
h=(xn-x0)/n
# Массив абсцисс
x=zeros(n+1)
# Массив для хранения точного решениия
yt=zeros(n+1)
# Цикл для формирования точного решения
for i=1:n+1
    x[i]=x0+(i-1)*h
    yt[i]=4*atan(x[i])
end
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(n+1)
ye[1]=y0
# Цикл для формирования решения методом Эйлера
for i=2:n+1
    ye[i]=ye[i-1]+h*f(x[i-1],ye[i-1])
end
# Реализация метода Рунге-Кутта
# Массив для хранения решения методом Рунге-Кутта
yrk=zeros(n+1)
yrk[1]=y0
# Цикл для формирования решения методом Рунге-Кутта
for i=2:n+1
    K1=f(x[i-1],yrk[i-1]);
    K2=f(x[i-1]+h/2,yrk[i-1]+h/2*K1);
```

```
    K3=f(x[i-1]+h/2,yrk[i-1]+h/2*K2);
    K4=f(x[i-1]+h,yrk[i-1]+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    yrk[i]=yrk[i-1]+delt;
 end
# Реализация метода Адамса
# Массив для хранения решения методом Адамса
ya=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ya[i]=yrk[i]
end
# Цикл для формирования решения методом Адамса
for i=5:n+1
    yp=ya[i-1]+h/24*(-9*f(x[i-4],ya[i-4])+37*f(x[i-3],ya[i-3])-59*f(x[i-2],y
    ya[i]=ya[i-1]+h/24*(f(x[i-3],ya[i-3])-5*f(x[i-2],ya[i-2])+19*f(x[i-1],ya
end
# Реализация метода Милна
# Массив для хранения решения методом Милна
ym=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ym[i]=yrk[i]
end
# Цикл для формирования решения методом Милна
for i=5:n+1
    yp=ym[i-4]+4*h/3*(2*f(x[i-3],ym[i-3])-f(x[i-2],ym[i-2])+2*f(x[i-1],ym[i-
    ym[i]=ym[i-2]+h/3*(f(x[i-2],ym[i-2])+4*f(x[i-1],ym[i-1])+f(x[i],yp));
end
# Вывод числа pi, которое получено путем численного решения оду y'=y разными
    @printf("x=%8.2f, ye=%8.6f, yrk=%8.6f, ya=%8.6f, ym=%8.6f, yt=%8.6f\n",
```

```
x=    1.00, ye=3.239926, yrk=3.141593, ya=3.141553, ym=3.141584, yt=3.141593
```

Задача 4.

$$y' = \frac{1}{x}, y(1) = 0.$$

Точное решение этого уравнение имеет вид $y = lnx$. Решая уравнение численно методами **Эйлера**, **Рунге-Кутта**, **Милна** и **Адамса** в точке x=1 должны получить число $ln2 = 0.693...$. Решение приведено ниже.

In [10]:
```
# Правая часть диференциального уравнения
f(x,y)=1/x
# Начальное условие
x0=Float64(1)
y0=Float64(0)
# Количество участков на интервале интегрирования
n=Int64(10)
# Правая граница интервала интегрирования
xn=Float64(2)
# Шаг интегрирования
h=(xn-x0)/n
# Массив абсцисс
```

```
x=zeros(n+1)
# Массив для хранения точного решениия
yt=zeros(n+1)
# Цикл для формирования точного решения
for i=1:n+1
    x[i]=x0+(i-1)*h
    yt[i]=log(x[i])
end
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(n+1)
ye[1]=y0
# Цикл для формирования решения методом Эйлера
for i=2:n+1
    ye[i]=ye[i-1]+h*f(x[i-1],ye[i-1])
end
# Реализация метода Рунге-Кутта
# Массив для хранения решения методом Рунге-Кутта
yrk=zeros(n+1)
yrk[1]=y0
# Цикл для формирования решения методом Рунге-Кутта
for i=2:n+1
    K1=f(x[i-1],yrk[i-1]);
    K2=f(x[i-1]+h/2,yrk[i-1]+h/2*K1);
    K3=f(x[i-1]+h/2,yrk[i-1]+h/2*K2);
    K4=f(x[i-1]+h,yrk[i-1]+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    yrk[i]=yrk[i-1]+delt;
 end
# Реализация метода Адамса
# Массив для хранения решения методом Адамса
ya=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ya[i]=yrk[i]
end
# Цикл для формирования решения методом Адамса
for i=5:n+1
    yp=ya[i-1]+h/24*(-9*f(x[i-4],ya[i-4])+37*f(x[i-3],ya[i-3])-59*f(x[i-2],y
    ya[i]=ya[i-1]+h/24*(f(x[i-3],ya[i-3])-5*f(x[i-2],ya[i-2])+19*f(x[i-1],ya
end
# Реализация метода Милна
# Массив для хранения решения методом Милна
ym=zeros(n+1)
# Формирование первых четырёх точек методом Рунге-Кутта
for i=1:4
    ym[i]=yrk[i]
end
# Цикл для формирования решения методом Милна
for i=5:n+1
    yp=ym[i-4]+4*h/3*(2*f(x[i-3],ym[i-3])-f(x[i-2],ym[i-2])+2*f(x[i-1],ym[i-
    ym[i]=ym[i-2]+h/3*(f(x[i-2],ym[i-2])+4*f(x[i-1],ym[i-1])+f(x[i],yp));
end
# Вывод ln2, которое получено путем численного решения оду y'=y разными мето
    @printf("x=%8.2f, ye=%8.6f, yrk=%8.6f, ya=%8.6f, ym=%8.6f, yt=%8.6f\n",
```

```
x=    2.00, ye=0.718771, yrk=0.693147, ya=0.693153, ym=0.693149, yt=0.693147
```

Задача 5 (об остывании кофе). Дифференциальное уравнение, описывающее процесс остывания кофе в чашке, известно в науке, как закон теплопроводности Ньютона.

$$\frac{dT}{dt} = -r \cdot (T - T_s), T(0) = T_0 = 90$$

здесь $T(t)$ - функция определяющая температуру чашки T в момент времени t, $T_s = 23$ - температура окружающей среды, r - коэффициент остывания, для керамической чашки 250 мл - $r = 0.035 \frac{\text{Вт}}{\text{м}\cdot\text{К}}$, для керамической чашки 125 мл - $r = 0.047 \frac{\text{Вт}}{\text{м}\cdot\text{К}}$. Точное решение имеет вид:

$$T(t) = T_0 + (T_s - T_0) \cdot e^{-rt}$$

Решим уравнение численно методами **Эйлера**, **Рунге-Кутта**. Реализация на языке *Julia* приведена ниже.

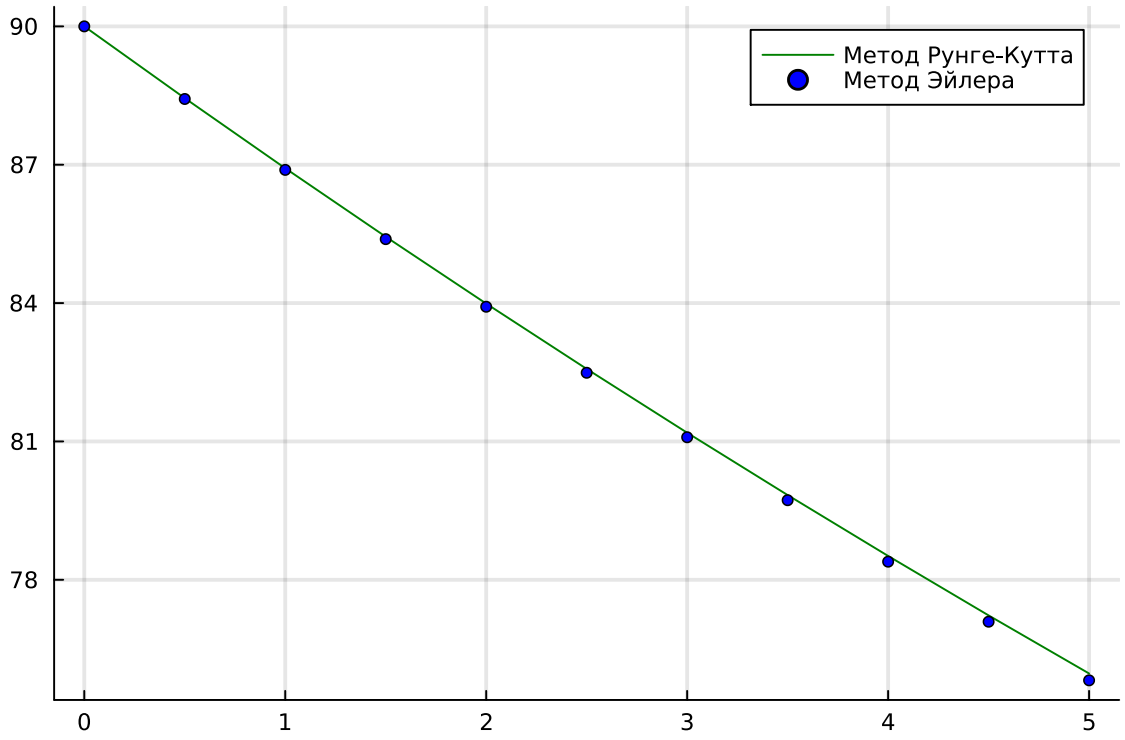
In [11]:
```
r=0.047
Ts=23
T0=90
# Правая часть диференциального уравнения
f(x,y)=-r*(y-Ts)
# Начальное условие
x0=Float64(0)
y0=Float64(90)
# Количество участков на интервале интегрирования
n=Int64(10)
# Правая граница интервала интегрирования
xn=Float64(5)
# Шаг интегрирования
h=(xn-x0)/n
# Массив абсцисс
x=zeros(n+1)
for i=1:n+1
    x[i]=x0+(i-1)*h
end
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(n+1)
ye[1]=y0
# Цикл для формирования решения методом Эйлера
for i=2:n+1
    ye[i]=ye[i-1]+h*f(x[i-1],ye[i-1])
end
# Реализация метода Рунге-Кутта
# Массив для хранения решения методом Рунге-Кутта
yrk=zeros(n+1)
yrk[1]=y0
# Цикл для формирования решения методом Рунге-Кутта
for i=2:n+1
```

```
    K1=f(x[i-1],yrk[i-1]);
    K2=f(x[i-1]+h/2,yrk[i-1]+h/2*K1);
    K3=f(x[i-1]+h/2,yrk[i-1]+h/2*K2);
    K4=f(x[i-1]+h,yrk[i-1]+h*K3);
    delt=h/6*(K1+2*K2+2*K3+K4);
    yrk[i]=yrk[i-1]+delt;
 end
# Цикл для вывода результатов
for i=1:n+1
    @printf("t=%8.2f, ye=%8.4f, yrk=%8.4f\n", x[i], ye[i], yrk[i])
end
```

```
t=    0.00, ye= 90.0000, yrk= 90.0000
t=    0.50, ye= 88.4255, yrk= 88.4439
t=    1.00, ye= 86.8880, yrk= 86.9239
t=    1.50, ye= 85.3866, yrk= 85.4392
t=    2.00, ye= 83.9205, yrk= 83.9889
t=    2.50, ye= 82.4889, yrk= 82.5724
t=    3.00, ye= 81.0909, yrk= 81.1888
t=    3.50, ye= 79.7258, yrk= 79.8373
t=    4.00, ye= 78.3927, yrk= 78.5172
t=    4.50, ye= 77.0910, yrk= 77.2277
t=    5.00, ye= 75.8199, yrk= 75.9682
```

In [12]:
```
# Графическое решение задачи
#plot(x, yt, color = :red, linewidth = 2, label = "точное решение",gridlinew
Plots.plot(x, yrk, color = :green, markersize = 3, label = "Метод Рунге-Кутт
Plots.scatter!(x, ye, color = :blue, markersize = 3, label = "Метод Эйлера")
```

Out[12]:



16.6. Использование пакета DifferentialEquations для решения

дифференциальных уравнений

Необходимо установить пакет DifferentialEquations

```
add DifferentialEquations
```

Потом его необходимо подключить

```
using DifferentialEquations
```

Рассмотрим использование пакета DifferentialEquations на примере решения следующей задачи Коши

Существует описание на русском
https://engee.com/helpcenter/stable/ru/julia/DifferentialEquations/getting_started.ht

Решить задачу Коши

$$\frac{dx}{dt} = 6\dot{x} - 13 \cdot t^3 - 22 \cdot t^2 + 17 \cdot t - 11 + sin(t)$$

$$x(0) = 2$$

Точное решение имеет вид

$$x(t) = \frac{119}{296} \cdot e^{6t} + \frac{1}{24} \cdot \left(52 \cdot t^3 + 114 \cdot t^2 - 30 \cdot t + 39\right) - \frac{1}{37} \cdot (6 \cdot \sin\ t + \cos$$

In [15]:
```
using DifferentialEquations
using Plots
#Оперелим дифференциальное уравнение
#Правая часть дифференциального уравнения
f(x, p, t) = 6*x-13*t^3-22*t^2+17*t-11+sin(t)
#f(x, p, t) = p[1]*x+p[2]*t^3+p[3]*t^2+p[4]*t+p[4]+sin(t)
#Начальное значение
u0 = 2
# Интервал интегрирования
tspan = (0.0, 1.0)
#p=[6 -13 -22 17 -11]
# Определяем всю задачу Коши
prob = ODEProblem(f, u0, tspan)
#prob = ODEProblem(f, u0, tspan,p)
```

Out[15]:
```
ODEProblem with uType Int64 and tType Float64. In-place: false
timespan: (0.0, 1.0)
u0: 2
```

In [16]:
```
sol = solve(prob, Tsit5(), reltol = 1e-8, abstol = 1e-8,saveat=0.1)
#sol = solve(prob, Tsit5(), reltol = 1e-8, abstol = 1e-8,saveat=0.1,save_eve
# Параметры решателя solve
```

```
#prob - сама задача,
# Tsit5() - метод решения - в данном случае метод Цитураса 5 поряка
# могут быть другие значения
# см. описнаия ниже , пречисленные ниже
# reltol - относительная точность, abstol - абсолютная точность,
# save_everystep = false - сохранять или нет каждую точку найденного решения
```

Out[16]:
```
retcode: Success
Interpolation: 1st order linear
t: 11-element Vector{Float64}:
 0.0
 0.1
 0.2
 0.3
 0.4
 0.5
 0.6
 0.7
 0.8
 0.9
 1.0
u: 11-element Vector{Float64}:
   2.0
   2.239126463636708
   2.8584051347890638
   4.094379683876679
   6.3672390369370735
  10.4317989097424
  17.65260938604915
  30.50523495166459
  53.48964737230455
  94.79453714721119
 169.3298877806762
```

Вот некоторые распространенные алгоритмы для решения ОДУ в **DifferentialEquations.jl** (https://engee.com/helpcenter/stable/ru/julia/DifferentialEquations/getting_started.ht

- **AutoTsit5(Rosenbrock23())** подходит для решения как жестких, так и нежестких уравнений. Это неплохой алгоритм в том случае, если об уравнении ничего не известно.
- **AutoVern7(Rodas5())** подходит для эффективного решения как жестких, так и нежестких уравнений с высокой точностью.
- **Tsit5()** подходит для стандартных нежестких уравнений. Это первый алгоритм, который следует попробовать в большинстве случаев.
- **BS3()** подходит для быстрого решения нежестких уравнений с низкой точностью.
- **Vern7()** подходит для решения нежестких уравнений с высокой точностью.

- **Rodas4()** или **Rodas5()** подходят для небольших жестких уравнений с определенными в Julia типами, событиями и т. д.
- **KenCarp4()** или **TRBDF2()** подходят для решения жестких уравнений среднего размера (100—2000 ОДУ).
- **RadauIIA5()** подходит для решения жестких уравнений с очень высокой точностью.
- **QNDF()** подходит для решения больших жестких уравнений.

Анализ решения

- solve(x) -- значение решения в точке x
- solve[n] -- обращение к значению решения номер *n*
- solve.t[n] -- значение абсциссы в точке номер *n*

In [17]:
```
sol(0.15)
```

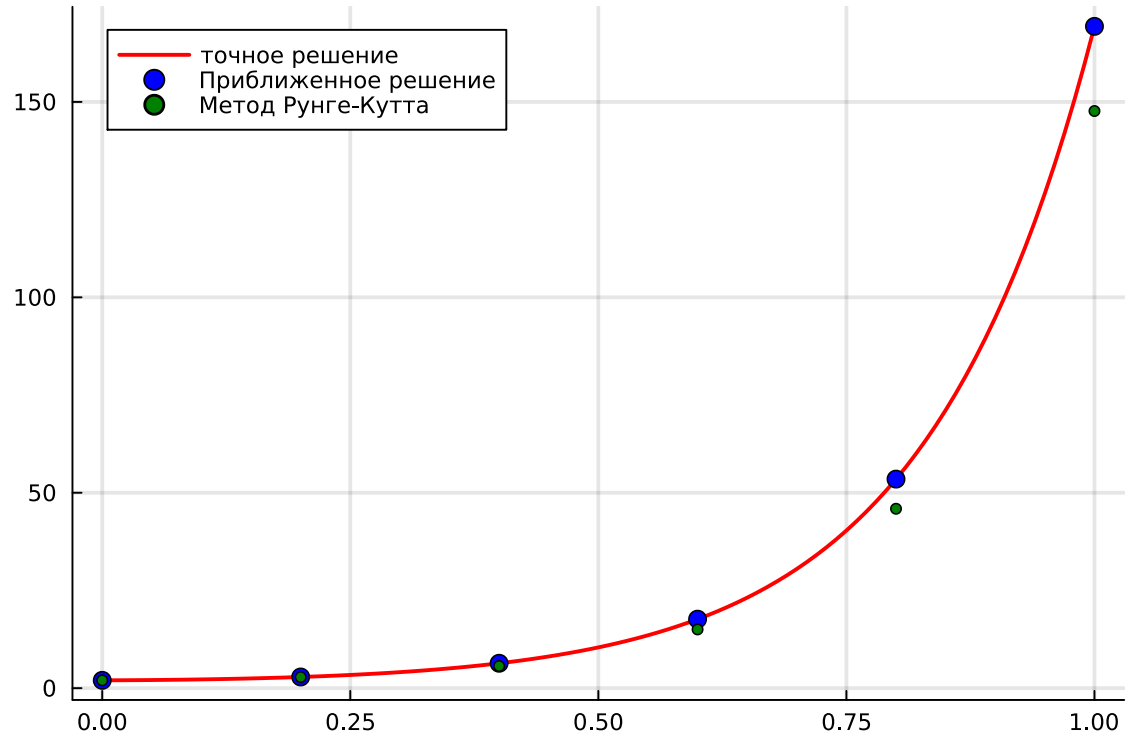
Out[17]:
```
2.548765799212886
```

In [18]:
```
g(t,x)=6*x-13*t^3-22*t^2+17*t-11+sin(t)
#точное решение
hx=0.01
n=Int64(1/hx)+1
tt=zeros(n)
yt=zeros(n)
for i=1:n
    tt[i]=(i-1)*hx
    yt[i]=119/296*exp(6*tt[i])+1/24 * (52 * tt[i]^3+114 * tt[i]^2- 30 * tt[i
    #y[i]=sol(tt[i])
end

#приближённое решение
h2=0.2
n2=Int64(1/h2)+1
t2=zeros(n2)
y2=zeros(n2)
yRK=zeros(n2)
for i=1:n2
    t2[i]=(i-1)*h2
    y2[i]=sol(t2[i])
end
#Метод Рунге-Кутта

yRK[1]=u0;
   for i=2:n2
    K1=g(tt[i-1],yRK[i-1]);
    K2=g(tt[i-1]+h2/2,yRK[i-1]+h2/2*K1);
    K3=g(tt[i-1]+h2/2,yRK[i-1]+h2/2*K2);
    K4=g(tt[i-1]+h2,yRK[i-1]+h2*K3);
    delt=h2/6*(K1+2*K2+2*K3+K4);
```

```
    yRK[i]=yRK[i-1]+delt;
  end
```

In [19]:
```
plot(tt, yt, color = :red, linewidth = 2, label = "точное решение",gridlinew
scatter!(t2, y2, color = :blue, markersize = 5, label = "Приближенное решени
scatter!(t2, yRK, color = :green, markersize = 3, label = "Метод Рунге-Кутта
```

Out[19]:



16.7. Решение системы обыкновенных дифференциальных уравнений

Найти решение задачи Коши для следующей системыдифференциальных уравнений:

$$\frac{dx}{dt} = y \cdot z$$

$$\frac{dy}{dt} = -x \cdot z$$

$$\frac{dz}{dt} = -\frac{1}{2} \cdot z \cdot y$$

$$x(0) = 0, y(0) = 1, z(0) = 1.$$

на интервале [0; 1].

Рассмотрим решение системы обыкновенных дифференциальных уравнений методом **Эйлера**.

In [20]:
```
# Правая часть системы
function fe(u,t)
   du=zeros(3)
   du[1]=u[2]*u[3];
   du[2]=-u[1]*u[3];
   du[3]=-u[1]*u[2]/2;
   return du
end
# Начальное условие
te0=Float64(0)
# Правая граница интервала интегрирования
ten=Float64(1)
ue0=[Float64(0);Float64(1);Float64(1)];
#println(u0);
# Количество участков на интервале интегрирования
ne=Int64(25)
# Шаг интегрирования
he=(ten-te0)/ne
# Массив абсцисс
te=zeros(ne+1)
for i=1:ne+1
    te[i]=te0+(i-1)*he
end
println(te)
# Реализация метода Эйлера
# Массив для хранения решения методом Эйлера
ye=zeros(3,ne+1)
ye[:,1]=ue0
for i=2:ne+1
    ye[:,i]=ye[:,i-1]+he*fe(ye[:,i-1],te[i-1])
end
#print(ye)
```

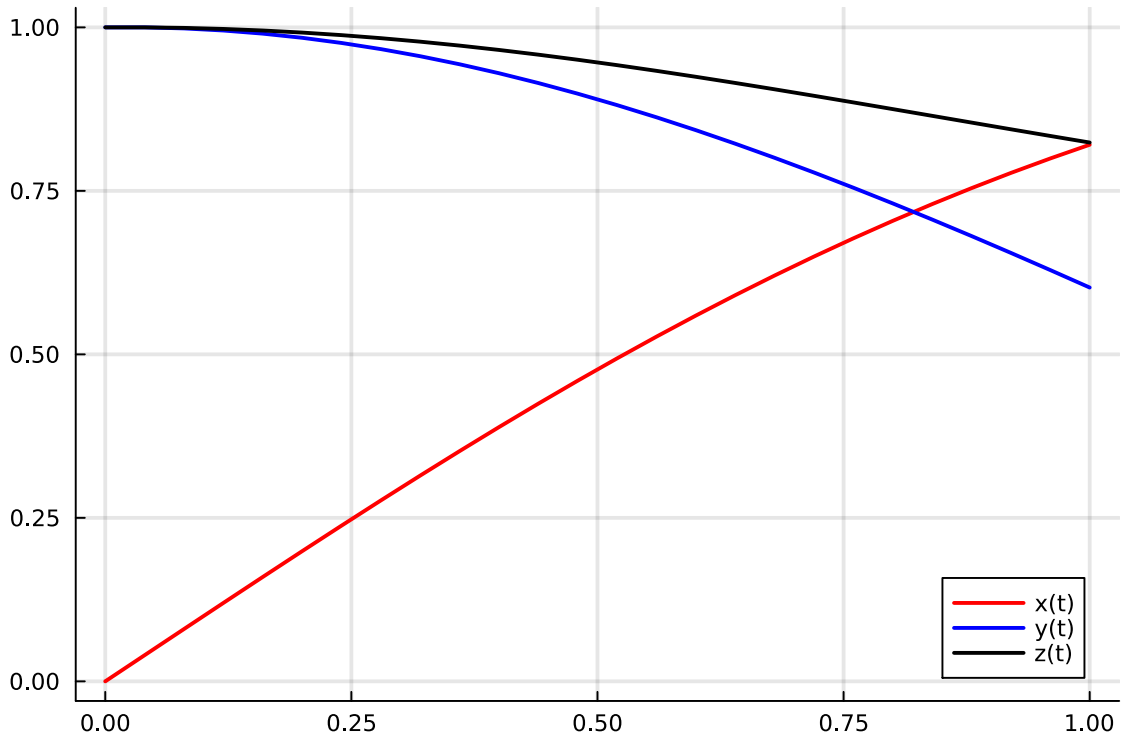
```
[0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.32, 0.36, 0.4, 0.44, 0.48,
0.52, 0.56, 0.6, 0.64, 0.68, 0.72, 0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0]
```

Самостоятельное задание. Реализуйте метод **Рунге-Кутта** для рассматриваемой системы.

График решения, найденного методом **Эйлера**.

In [21]:
```
Plots.plot(te, ye[1,:], color = :red, linewidth = 2, label = "x(t)",gridline
Plots.plot!(te, ye[2,:], color = :blue, linewidth = 2, label = "y(t)",gridli
Plots.plot!(te, ye[3,:], color = :black, linewidth = 2, label = "z(t)",gridl
```

Out[21]:



А теперь найдём решенние используя пакет DifferentialEquations

In [22]:
```
function f(du, u,t)
   du[1]=u[2]*u[3];
   du[2]=-u[1]*u[3];
   du[3]=-u[1]*u[2]/2;
end
```

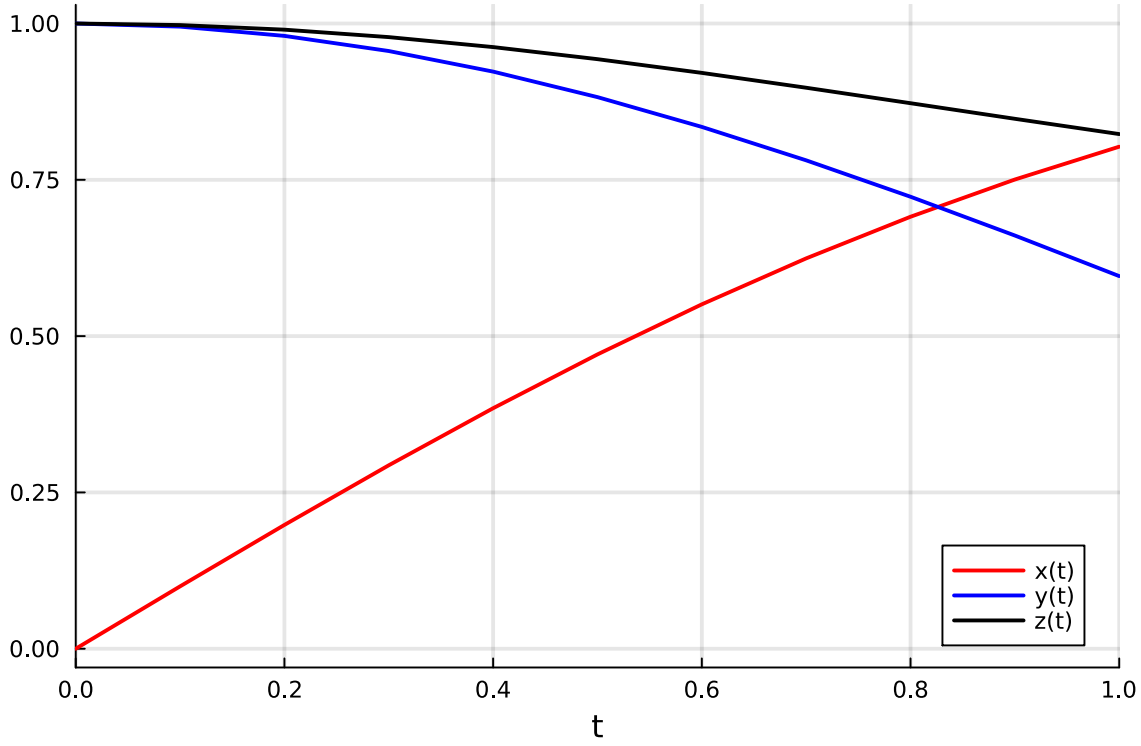
Out[22]:
```
f (generic function with 2 methods)
```

In [23]:
```
using DifferentialEquations
function syst1(du, u,p,t)
   du[1]=u[2]*u[3];
   du[2]=-u[1]*u[3];
   du[3]=-u[1]*u[2]/2;
end
u0 = [0; 1;1]
tspan = (0, 1)
prob = ODEProblem(syst1, u0, tspan)
sol = solve(prob,saveat=0.1)
#sol = solve(prob, AutoTsit5(Rosenbrock23()), reltol = 1e-8, abstol = 1e-8)
#sol = solve(prob, Rodas5(), reltol = 1e-8, abstol = 1e-8)
```

Out[23]:
```
retcode: Success
Interpolation: 1st order linear
t: 11-element Vector{Float64}:
 0.0
 0.1
 0.2
 0.3
 0.4
 0.5
 0.6
 0.7
 0.8
 0.9
 1.0
u: 11-element Vector{Vector{Float64}}:
 [0.0, 1.0, 1.0]
 [0.09975069692157078, 0.9950124574490465, 0.9975093446926175]
 [0.19802171304561336, 0.9801976504534762, 0.9901483348406972]
 [0.29341286099925135, 0.9559856807824527, 0.9782403667954936]
 [0.3846721134203797, 0.9230534381368, 0.9622961584497687]
 [0.4707503267853382, 0.882265808330789, 0.9429719914112777]
 [0.5508311685459317, 0.8346170176425881, 0.92102814879505]
 [0.6243399969713977, 0.7811526334906391, 0.8972734753476033]
 [0.6909341435255074, 0.7229165924795662, 0.8725273083774598]
 [0.7504783033475109, 0.660895315894316, 0.8475796793936312]
 [0.8030014185827887, 0.5959763779394863, 0.8231608250543999]
```

In [24]:
```
#Построение графика решения
using Plots
Plots.plot(sol, idxs=(0,1),  color = :red, linewidth = 2, label = "x(t)",gri
Plots.plot!(sol, idxs=(0,2) , color = :blue, linewidth = 2, label = "y(t)",g
Plots.plot!(sol, idxs=(0,3), color = :black, linewidth = 2, label = "z(t)",g
```

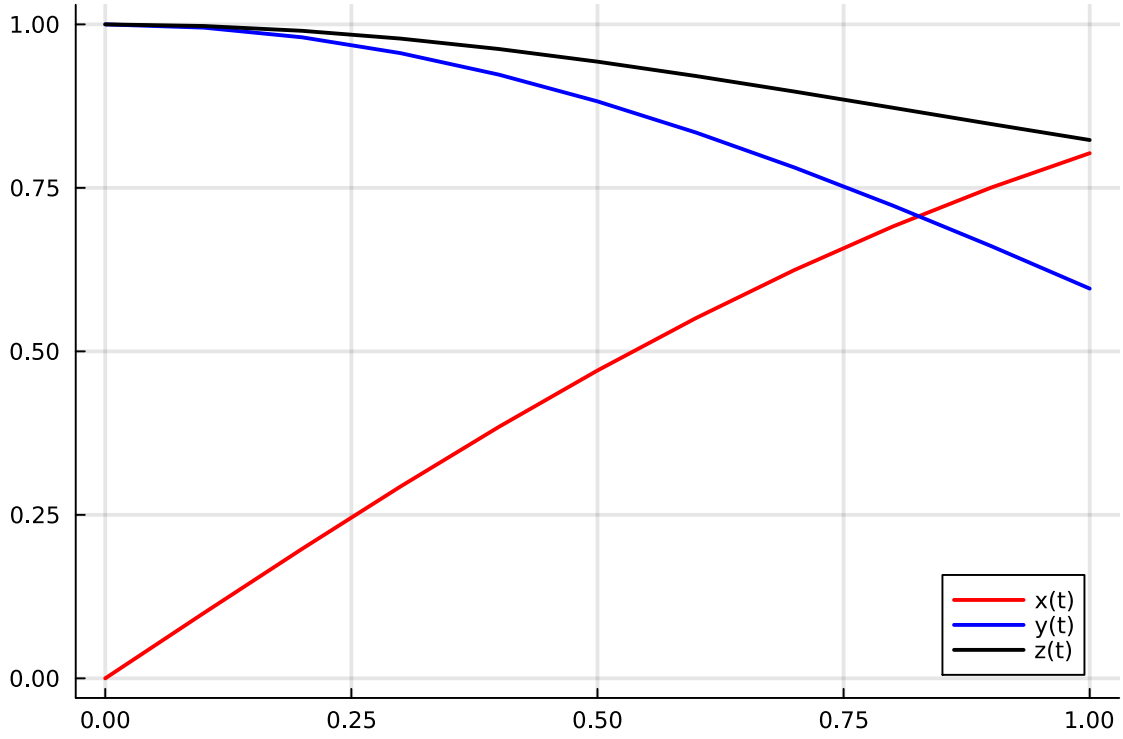
Out[24]:



In [25]:

```
#Другая версия построения графика решения

ttt=zeros(11)
x=zeros(11)
y=zeros(11)
z=zeros(11)
for i=1:11
    ttt[i]=sol.t[i]
    x[i]=sol.u[i][1]
    y[i]=sol.u[i][2]
    z[i]=sol.u[i][3]
end
Plots.plot(ttt, x, color = :red, linewidth = 2, label = "x(t)",gridlinewidth
Plots.plot!(ttt, y , color = :blue, linewidth = 2, label = "y(t)",gridlinewi
Plots.plot!(ttt, z, color = :black, linewidth = 2, label = "z(t)",gridlinewi
```

Out[25]:

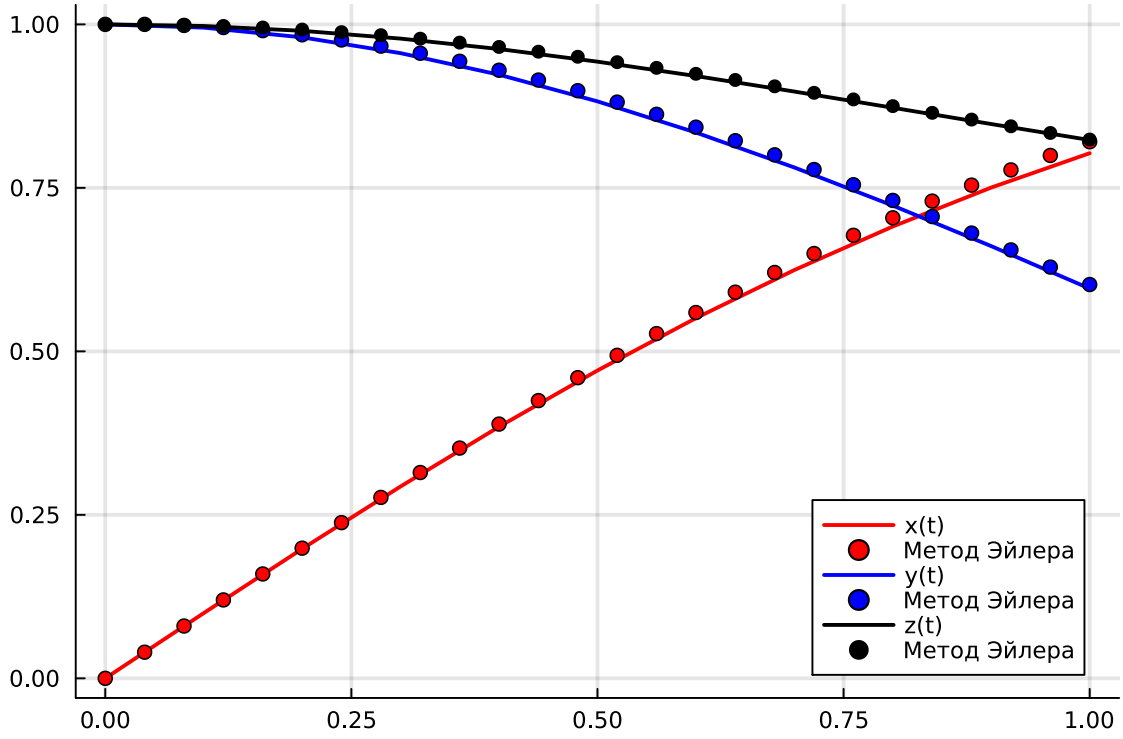


Наложим точки, найденные с помощью метода **Эйлера** на график решения, полученный с помощью ***solve***.

In [26]:

```
Plots.plot(ttt, x, color = :red, linewidth = 2, label = "x(t)",gridlinewidth
Plots.scatter!(te,ye[1,:] , color = :red, markersize = 4, label = "Метод Эйл
Plots.plot!(ttt, y , color = :blue, linewidth = 2, label = "y(t)",gridlinewi
Plots.scatter!(te,ye[2,:] , color = :blue, markersize = 4, label = "Метод Эй
Plots.plot!(ttt, z, color = :black, linewidth = 2, label = "z(t)",gridlinewi
Plots.scatter!(te,ye[3,:] , color = :black, markersize = 4, label = "Метод Э
```

Out[26]:



Глава 17. Численное решение уравнений в частных производных

Здесь на примере параболичского уравнения рассмотрим явную разностную схему численного решения уравнений в частных производных, на примере эллиптического уравнения - неявную разностную схему решения уравнения в частных производных.

Краткое описание разностных методов решения уравнений в частных производных можно найти в книге билиотеки ALT - **Е.Р.Алексеев, К.В. Дога, О.В. Чеснокова Scilab: Решение инженерных и математических задач** https://www.altlinux.org/Images.www.altlinux.org/3/3a/Scilab.pdf. Для более подробного изучения рекомендуем учебники **Самарский А.А. Теория разностных схем**, **Вержбицкий В.М. Основы численных методов**.

Рассмотрим решение параболического уравнения на примере следующей задачи.

Задача 1. Решить параболическое уравнение, описывающее распределение температуры в стержне длиной L, начальная температура стержня задается произвольной функцией $\varphi(x)$. Температуры концов стержня равны $u(0,t) = U_1 = \text{const}$, $u(L,t) = U_2 = \text{const}$

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial t^2} + f(x,t), \qquad a^2 = \frac{\lambda}{c\rho}, \qquad 0 < x < L, \qquad 0 < t < \infty,$$
$$u(0,t) = U_1 = \text{const}, \qquad u(L,t) = U_2 = \text{const}, \qquad 0 < t < \infty,$$
$$u(x,0) = \varphi(x), \qquad 0 < x < L,$$

здесь a^2 - коэффициент температуропроводности, λ- коэффициент теплопроводности материала стержня, c - удельная теплоемкость, ρ - плотность.

Заменим производные разностными соотношениями

$$\frac{\partial u(x_i, t_j)}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta}$$
$$\frac{\partial^2 u(x_i, t_j)}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}.$$

Получим явную разностную вычислительную схему

$$u_{i,j+1} = \gamma \cdot u_{i,j-1} + (1 - 2\gamma) \cdot u_{i,j} + \gamma u_{i,j+1} + \Delta \cdot f_{i,j}$$
$$u_{0,j} = \mu_j, \qquad u_{n,j} = \nu_j, \qquad u_{i,0} = \varphi_i, \qquad \gamma = \frac{a^2 \Delta}{h^2}$$

Здесь h - шаг по переменной x, $h = \frac{L}{h}$, $i = 0, 1, 2, \ldots, n$, Δ - шаг по времени, $\Delta = \frac{t_k - t_0}{k}$, $j = 0, 1, \ldots k$.

Ниже приведён код на языке Julia решения задачи 1.

Входными данными программы решения задачи являются:

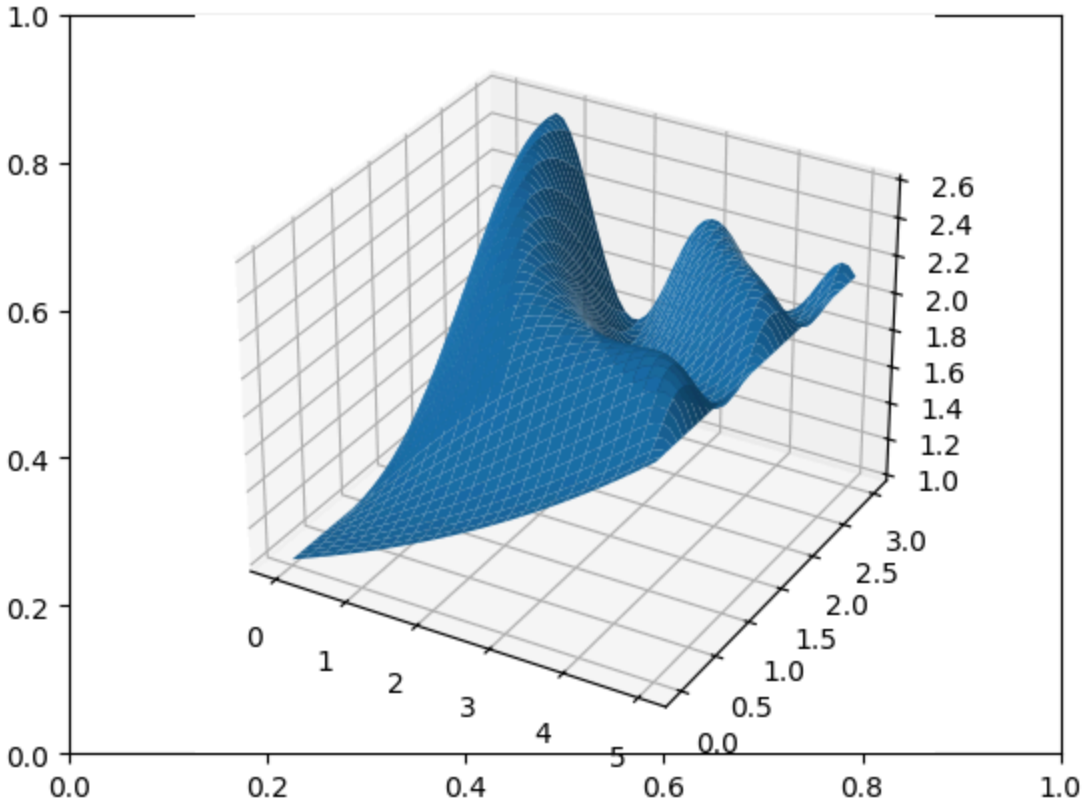
1. N=50 - количество интервалов, на которые разбивается отрезок $(0, L)$.
2. K=200 - количество интервалов, на которые разбивается отрезок $(0, T)$.
3. L=5 - длина стержня.
4. T=3 - интервал времени.
5. a=0.4 - параметр дифференциального уравнения.

Результат:

1. Решение - сеточная функция u, определенная на сетке Ω_h^Δ.
2. Массивы x и t.

In [28]:
```
using PyPlot
#using Plots
#Правая часть дифференциального уравнения.
function f(x,t)
```

```
  return sin(x*t)
end
#Начальное условие
function fi(x)
  return exp(0.15*x)
end
#Условие на левой границе
function myu(t)
  return 1
end
#Условие на правой границе
function nyu(x)
  return 2.117
end
#Решение параболического уравнения методом сеток с
#помощью явной разностной схемы. N - количество участков,
#на которые разбивается интервал по x (0,L); K - количество
#участков, на которые разбивается интервал по t (0,T); a -
#параметр дифференциального уравнения теплопроводности,
#Функция возвращает матрицу решений u и вектора x, t
#Вычисляем шаг по x
N=50
K=200
L=5
T=3
a=0.4
#Вычисляем шаг по x
h=L/N;
#Вычисляем шаг по t
  delta=T/K;
#Формируем массив x и первый столбец матрицы решений U
#из начального условия
x=zeros(N+1)
t=zeros(K+1)
u=zeros(N+1,K+1)
for i=1:N+1
    x[i]=(i-1)*h;
    u[i,1]=fi(x[i]);
end
#Формируем массив t, первую и последнюю строку
#матрицы решений U из граничных условий
for j=1:K+1
    t[j]=(j-1)*delta;
    u[1,j]=myu(t[j]);
    u[N+1,j]=nyu(t[j]);
end
gam=a^2*delta/h^2;
#Формируем матрицу решений u с помощью явной разностной схемы
for j=1:K
    for i=2:N
        u[i,j+1]=gam*u[i-1,j]+(1-2*gam)*u[i,j]+gam*u[i+1,j]+delta*f(x[i],t[j
    end
  end
# Строим график решения
# Вывод  решения u с помощью функции surf из пакета PyPlot
PyPlot.surf(x,t,u')
```



Out[28]: PyObject <mpl_toolkits.mplot3d.art3d.Poly3DCollection object at 0x7fb8205cb1c0>

Задача 2. Рассмотрим разностную схему для решения эллиптического уравнения

$$\Delta u = \frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} - \frac{5}{x}\frac{\partial \Psi}{\partial x} = -2$$

в прямоугольной области

$$\Omega(R - b \leqslant x \leqslant R + b, -a \leqslant y \leqslant a)$$

с граничными условиями Дирихле на границе Γ

$$\Psi_{(x,y)\in\Gamma} = 0.$$

Построим сетку Ω_{hx}^{hy}, для чего проведем в области Ω прямые, параллельные осям $y = y_j$ и $x = x_i$, где $x_i = R - b + i \cdot hx$, $hx = \frac{2b}{Nx}$, $i = 0, 1, 2, \ldots, Nx$, $y_j = -a + j \cdot hy$, $hy = \frac{2a}{Ny}$, $j = 0, 1, \ldots, Ny$. Для построения разностного уравнения заменим частные производные и граничные условия следующими соотношениями:

$$\frac{\partial^2\Psi(x_i,y_j)}{\partial x^2}=\frac{\Psi_{i-1,j}-2\Psi_{i,j}+\Psi_{i+1,j}}{hx^2}$$
$$\frac{\partial^2\Psi(x_i,y_j)}{\partial y^2}=\frac{\Psi_{i,j-1}-2\Psi_{i,j}+\Psi_{i,j+1}}{hy^2}$$

$$\Psi_{i,0}=\Psi_{i,Ny}=0,\qquad i=0,1,\ldots,Nx$$
$$\Psi_{0,j}=\Psi_{Nx,j}=0,\qquad j=0,1,\ldots,Ny$$

С помощью приведённых выше соотношений преобразуем эллиптическую краевую задачу к следующей системе разностных уравнений.

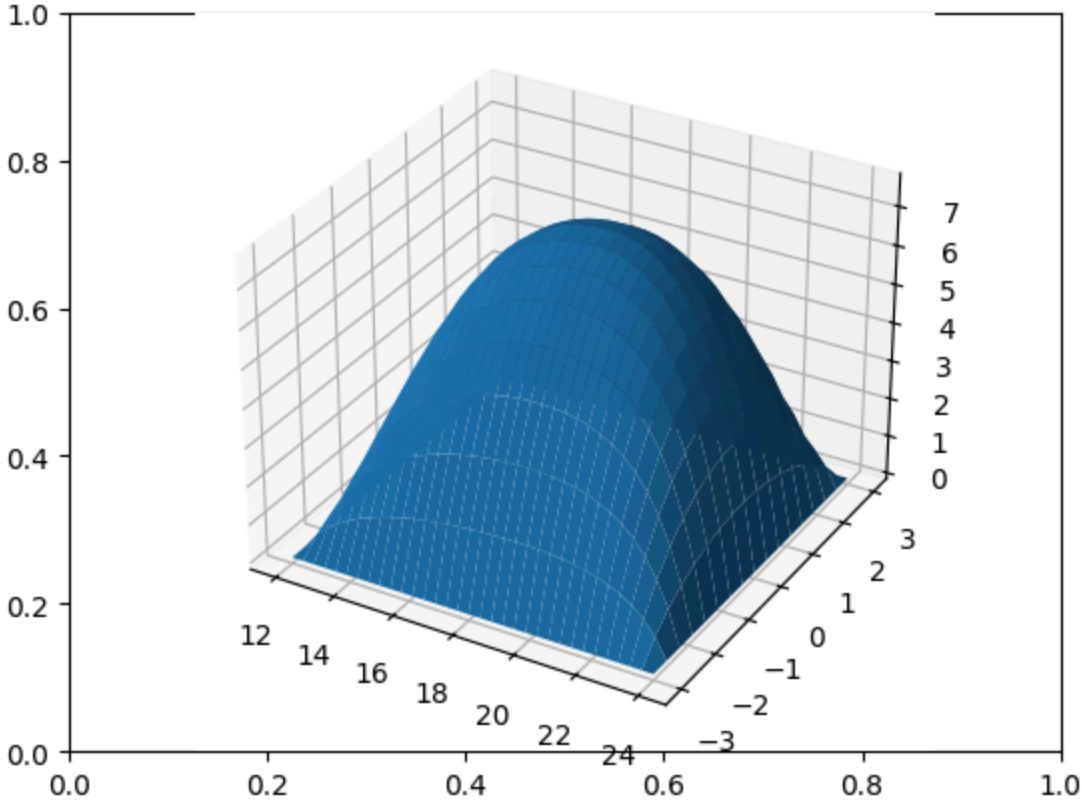
$$\Psi_{i,j}=\frac{1}{A}\left(B_i\Psi_{i+1,j}+C_i\Psi_{i-1,j}+D(\Psi_{i,j-1}+\Psi_{i,j+1})+2\right)$$

$$A=\frac{2}{hx^2}+\frac{2}{hy^2},\qquad B_i=\frac{1}{hx^2}+\frac{5}{2hx\,x_i},\qquad C_i=\frac{1}{hx^2}-\frac{5}{2hx\,x_i},\qquad D$$
$$i=1,2,\ldots,Nx-1;\qquad j=1,2,\ldots,Ny-1$$
$$\Psi_{i,0}=\Psi_{i,Ny}=0,\qquad i=0,1,\ldots,Nx$$
$$\Psi_{0,j}=\Psi_{Nx,j}=0,\qquad i=0,1,\ldots,Ny$$

Эту систему можно решать итерационными методами (например **методом Зейделя**). В случае медленной сходимости итерационных процессов при решении сеточных уравнений, получаемых при аппроксимации гиперболических и эллиптических задач, имеет смысл попробовать заменить метод Зейделя градиентными методами (или методами релаксации). На листинге представлено решение уравнения сеточным методом, а также - график найденного решения.

In [29]:
```
using PyPlot
r=18
a=3
b=6
Nx=32
Ny=16
eps=0.01
#Входные данные:
#r, a, b - значения, определяющие область решения задачи,
#Nx - количество участков, на которые разбивается интервал по
#x(R-b,R+b);
#Ny - количество участков, на которые разбивается интервал по
#y (-a,a);
#eps - точность решения уравнения методом Зейделя.
#Выходные данные:
#psi - матрица решений в узлах сетки, массив x, массив y,
#k - количество итерация при решении разностного уравнения
# методом Зейделя.
#Вычисляем шаг по y
hy=2*a/Ny;
```

```
#Вычисляем шаг по x
hx=2*b/Nx;
x=zeros(Nx+1)
B=zeros(Nx+1)
C=zeros(Nx+1)
y=zeros(Ny+1)
psi=zeros(Nx+1,Ny+1)
R=zeros(Nx,Ny)
#Формируем массив x, первый и последний столбцы матрицы
#решений psi из граничного условия
for i=1:Nx+1
    x[i]=r-b+(i-1)*hx;
    psi[i,1]=0;
    psi[i,Ny+1]=0;
end
#Формируем массив y, первую и последнюю строки матрицы
#решений psi из граничного условия
for j=1:Ny+1
    y[j]=-a+(j-1)*hy;
    psi[1,j]=0;
    psi[Nx+1,1]=0;
end;
#Вычисляем коэффициенты разностного уравнения
A=2/hy^2+2/hx^2;
D=1/hy^2;
for i=2:Nx+1
    B[i]=1/hx^2+5/(2*hx*x[i]);
    C[i]=1/hx^2-5/(2*hx*x[i]);
end
#Решение разностного уравнения методом Зейделя с
#точностью eps
p=1;
k=0;
while p>eps
# Формируем очередное приближение - матрицу psi и матрицу ошибок R
    for i=2:Nx
      for j=2:Ny
        V=1/A*(B[i]*psi[i-1,j]+C[i]*psi[i+1,j]+D*(psi[i,j-1]+psi[i,j+1])+2);
        R[i,j]=abs(V-psi[i,j]);
        psi[i,j]=V;
      end
    end
 # Вычисляем норму матрицы ошибок
    p=R[2,2];
    for i=2:Nx
      for j=2:Ny
        if R[i,j]>p
          p=R[i,j];
        end
      end
    end
  k=k+1;
end
# Вывод  решения psi с помощью функции surf из пакета PyPlot
PyPlot.surf(x,y,psi');
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
12
14
16
18
20
22
24
−3
−2
−1
0
1
2
3
0
1
2
3
4
5
6
7

In [ ]: