Programming in Julia

Nathan Warner

Computer Science
Northern Illinois University
United States

Contents

Creating and running Julia scripts, and outputting

First, make sure you have Julia (juliaup) downloaded on your machine. Create a .jl file and run it with

```
julia script.jl
```

Here is a simple Hello World! script written in julia

```
# helloworld.jl
println("Hello World!")
```

1.1 Outputting data

1.1.1 println

Prints the value followed by a newline. Suitable for general-purpose printing

```
println("Hello world")
```

1.1.2 Print

Similar to `println`, but does not append a newline

1.1.3 Formatted output: @printf from the Printf module

Prints the value followed by a newline. Suitable for general-purpose printing

```
using Printf
    @printf("Pi to 3 decimal places: %.3f\n", pi)  # Output: Pi
↪   to 3 decimal places: 3.142
```

1.1.4 Error and debugging output

- **@warn**: Logs a warning message.

```julia
@warn "This is a warning message."
```

- **@info**: Logs an informational message.

```julia
@info "This is an informational message."
```

- **@error**: Logs an error message.

```julia
@error "This is an error message."
```

- **@show**: Prints the expression and its value. Useful for debugging.

```julia
x = 42
@show x  # Output: x = 42
```

1.1.5 Display

Displays a value using a richer representation (e.g., for plots or tables in Jupyter).

```julia
display("Hello, World!")  # Output: "Hello, World!"
```

The basics

2.1 Arithmetic operators

Julia supports the following arithmetic operators

- **Addition:** `+` (e.g., `a + b`)
- **Subtraction:** `-` (e.g., `a - b`)
- **Multiplication:** `*` (e.g., `a * b`)
- **Division:** `/` (e.g., `a / b`)
- **Integer Division:** `div` (e.g., `div(a, b)`)
- **Modulo (Remainder):** `%` (e.g., `a % b`)
- **Floor Division:** `//` (e.g., `a // b`)
- **Power:** ^ (e.g., a^b)
- **Negation:** `-` (e.g., `-a`)

2.2 Data types

Julia has the following data types

2.2.1 Numerical Types

- **Integers:**
  - `Int8, Int16, Int32, Int64, Int128:` Signed integers with various bit sizes.
  - `UInt8, UInt16, UInt32, UInt64, UInt128:` Unsigned integers.
  - `Int:` Default signed integer type (dependent on the platform, typically Int64 or Int32).
- **Floating-point numbers:** `Float16, Float32, Float64:` IEEE 754 floating-point numbers.
- **Big numbers:**
  - `BigInt:` Arbitrary precision integers.
  - `BigFloat:` Arbitrary precision floating-point numbers.
- **Complex numbers:** `Complex{T}:` Complex numbers with real and imaginary parts of type T.
- **Rational numbers:** `Rational{T}:` Fractions represented as numerator//denominator.

In Julia, the default type for a literal integer like 15 is Int (platform-dependent, typically Int64 on 64-bit systems). However, you can explicitly create integers of specific types (Int8, Int16, etc.) using constructors. For example,

```julia
x = Int8(15)  # Creates an Int8 with value 15
```

2.2.2 Boolean Type

- **Bool**: true or false

2.2.3 Characters and Strings

- **Char:** Single Unicode character (e.g., ’a’).
- **String:** A sequence of characters (e.g., ”Hello, world!”).

2.2.4 Abstract Data Types

- **Number:** Abstract type for all numbers.
- **Real:** Abstract type for real numbers (Int, Float64, etc.).
- **AbstractString:** Abstract type for string-like objects.

2.2.5 Composite Types

- **Tuples:** Fixed-size collections of values, e.g., (1, ”hello”, true).
- **NamedTuples:** Tuples with named fields, e.g., (a=1, b=2).

2.2.6 Collection Types

- **Arrays**:
  - **1D arrays (vectors):** Vector{T} (e.g., [1, 2, 3]).
  - **2D arrays (matrices):** Matrix{T} (e.g., [1 2; 3 4]).
  - **Higher-dimensional arrays:** Array{T, N}.
- **Ranges:**
  - **1:10:** A range from 1 to 10.
  - **1:2:10:** A range with a step of 2.
- **Dictionaries:**
  - **Dict{K, V}:** A collection of key-value pairs, e.g., Dict(”a” => 1, ”b” => 2).
- **Sets**:
  - **SetT:** An unordered collection of unique elements, e.g., Set([1, 2, 3])

2.2.7 Nothing and Missing:

- **Nothing:** Represents the absence of a value (similar to null in other languages).
- **Missing:** Represents missing data (useful in data analysis).

2.2.8 User-defined Types

- **struct:** Immutable composite types.
- **mutable struct:** Mutable composite types.

2.2.9 typeof()

We can use the `typeof()` function to retrieve the type of a variable

```julia
x = 10
println(typeof(x))
```

2.3 Type conversions

2.3.1 Using Constructors

Julia uses constructors to convert a value to a specific type. This is the most common way to perform type casting.

```julia
# Convert to Integer Types
x = Int8(42)       # Converts 42 to an Int8
y = UInt16(300)    # Converts 300 to a UInt16
```

2.3.2 Using convert Function

The convert function explicitly converts a value to the desired type.

```julia
convert(Type, value)
```

For example

```julia
# Convert to Int
x = convert(Int, 42.5)       # Converts 42.5 to 42 (truncates the decimal)

# Convert to Float64
y = convert(Float64, 42)     # Converts 42 to 42.0

# Convert to String
s = convert(String, 123)     # Converts 123 to "123"
```

2.3.3 Parsing Strings

To convert a string to a numerical type, use the parse function.

```
parse(Type, string)
```

For example,

```
# Parse to Integer
x = parse(Int, "123")        # Converts "123" to 123

# Parse to Float64
y = parse(Float64, "3.14")   # Converts "3.14" to 3.14
```

2.3.4 Automatic Conversion

Some operations perform automatic type promotion or conversion.

```
a = 3          # Int
b = 2.5        # Float64
c = a + b      # Automatically promotes `a` to Float64
println(c)     # Output: 5.5
println(typeof(c))  # Output: Float64
```

2.4 Operator precedence

Julia follows PEMDAS

2.5 String Operations

In general, you can't perform mathematical operations on strings, even if the strings look like numbers. But there are two exceptions, * and ^.

The * operator performs string concatenation, which means it joins the strings by linking them end-to-end. For example:

The ^ operator also works on strings; it performs repetition. For example, ”Spam”^3 is ”SpamSpamSpam”. If one of the values is a string, the other has to be an integer.

2.6 Comments

Julia uses the pound sign for comments

```
# Comment
```

Functions